الجمهورية العربية السورية
وزارة التعليم العالي
جامعة البعث
كلية الهندسة المعلوماتية
قسم هندسة الشبكات والنظم الحاسوبية

محاكاة إطار المقابلة والاختزال

دراسة أعدت لنيل درجة الدكتوراه في هندسة النظم والشبكات الحاسوبية

إعداد: المهندس **عامر احمد دوار**

إشراف

الأستاذ الدكتور محمد النايف الحاج يونس والدكتور سهيل الحمود

العام الدراسي 2021/2020

كلمة شكر وتقدير

وصل بي قطار العلم إلى المحطة المرام بعد رحلة كانت تحمل في طياتها المشقة والتعب والأمل والإنجاز، ولا يسعني إلا أن أقدم أكبر آيات الشكر والعرفان للكادر التدريسي في كلية الهندسة المعلوماتية الذين قدموا لي التوجيه والنصح ولم يبخلوا على بالعلم والمعرفة وأخص بالشكر

المشرف: الأستاذ الدكتور المهندس محمد النايف الحاج يونس

المشرف المشارك: الدكتور المهندس سهيل الحمود

واللذان تفضلا مشكورين بالإشراف على هذا البحث.

جامعة البعث – كلية الهندسة المعلوماتية

م. عامر احمد دوار

الملخص

يعد إطار المقابلة والاختزال (Map–Reduce) نموذج برمجي تفرعي يستخدم لمعالجة مجموعات بيانية ضخمة. تم تأسيس إطار المقابلة والاختزال وتحقيقه مفتوح المصدر Hadoop ليكونا من أكثر أدوات تحليل البيانات استخداماً. إن هذا النموذج سهل الاستخدام ويعد بتقليل الوقت اللازم لمعالجة البيانات كما يمكن تشغيله على العتاد الصلب المتوفر. إن تجهيز عناقيد مقابلة واختزال ضخمة يستلزم الحذر في ضبط العديد من معاملات زمن التشغيل الضرورية للوصول إلى أداء فعال. إن العدد الكبير في متغيرات الإعداد الخاصة بـ Hadoop تشكل عدداً من التحديات للمستخدم، لأنه يصبح من الصعب على مستخدم Hadoop تحديد المتغيرات التي تساعده في الوصول إلى الأداء الجيد. كما أنه من الصعب إن لم يكن من المستحيل إعداد بيئة Hadoop فيزيائية من أجل اختبار قدرة تطبيقات Hadoop على التوسعية التي يمكن أن تصل إلى المئات وربما الآلاف من العقد. إن هذه التحديات تجعل من محاكيات إطار المقابلة والاختزال أمراً ضرورياً، بحيث يمكن استخدام المحاكيات من أجل ضبط أداء عناقيد المقابلة والاختزال، كما تساعد المحاكيات في دراسة سلوك تطبيقات المقابلة والاختزال. يقدم هذا البحث دراسة مرجعية وافية عن أشهر محاكيات إطار المقابلة والاختزال، تم في هذا البحث تقديم المحاكي HSG(Hadoop SimGrid) والذي صممناه بحيث يساعد على ضبط إعدادات Hadoop من أجل الوصول إلى الأداء الأمثل. اختبرنا دقة النتائج وقدرة المحاكي على التوسع ومحاكاة بنى عنقودية متنوعة حيث استطاع هذا المحاكي محاكات عناقيد حاسوبية مؤلفة من عدة آلاف من العقد وأعمال يصل حجمها إلى 20 TBوضمن زمن معقول. يتميز هذا المحاكي بسهولة الاستخدام من حيث ضبط الاعدادات من خلال ملفات من نوع JSON، كما يدعم عمليات القراءة والكتابة والتكرار على HDFS بشكل مفصل وهذا ما لم يتم لحظه بهذا التفصيل في باقي المحاكيات.

الكلمات المفتاحية: النظم الموزعة، الحوسبة السحابية، محاكاة الشبكات الحاسوبية، البيانات الضخمة.

ABSTRACT

Map-Reduce is a parallel programming model to process large datasets. The Map-Reduce framework and its open source implementations including Hadoop have established themselves as one of the most popular large data sets analyzers. The model is easy-to-use, and promising in reducing time-to-solution and can be run on commodity hardware. Setting up and operating a large Map-Reduce cluster entails careful evaluation of various design choices and run-time parameters to achieve high efficiency. However, the large number of configuration parameters of Hadoop brings forth a number of challenges to users. For a user Hadoop application, it is hard to decide on a set of parameters that would help to achieve a good performance. It would be extremely difficult if not impossible to set up a physical Hadoop environment to evaluate the scalability of a Hadoop application up to a few hundred or even thousand nodes. These challenges make it a necessity to have a map-reduce simulator in place where it can be used to tune the performance of a map-reduce cluster and to study the behaviors of Map-Reduce applications. In this paper, we introduce a detailed survey about the most famous map-reduce simulators. Then, a new Hadoop simulator "HSG Hadoop SimGrid" was designed and implemented completely from scratch to allow accurate configuration for Hadoop clusters to reach optimized performance. We tested the accuracy of the results and the simulator's ability to scale and simulate various cluster structures. This simulator was able to simulate computer clusters consisting of several thousand nodes and jobs up to 20TB in a reasonable time. This simulator is characterized by ease of use in terms of adjusting settings through JSON files, and supports reading, writing and replication operations on HDFS in detail, and this is not noticed in this detail in other simulator.

Keywords: Distributed Systems, Cloud Computing, Computer Network Simulation, Big Data.

الإنتاج العلمي

- بحث علمي في مجلة جامعة البعث بالمجلد **41** لعام **2019** بعنوان:

"تطوير أداة لمحاكاة نظام الملفات الموزع في إطار المقابلة والاختزال"

وذلك بالاشتراك **مع الأستاذ الدكتور محمد النايف الحاج يونس والدكتور سهيل الحمود**

- بحث علمي في مجلة جامعة البعث بالمجلد **41** لعام **2019** بعنوان:

"تطوير وتحقيق محاك لإطار المقابلة والاختزال"

وذلك بالاشتراك مع **الأستاذ الدكتور محمد النايف الحاج يونس والدكتور سهيل الحمود**

الفهرس

قائمة الأشكال والرسومات البيانية.....ص

قائمة الجداول ..... ظ

الفصل الأول- المقدمة والهدف من البحث.....1

1-1 المقدمة 1

1-3 مقارنة Hadoop مع باقي الأنظمة 3

1-4 دوافع البحث 5

1-5 الأهداف 6

1-6 المساهمة 6

1-7 اقسام الرسالة.....7

الفصل الثاني - المقابلة والاختزال 9

2-1 المقدمة.....9

2-2 آلية عمل إطار المقابلة والاختزال 9

2-2-1 تدفق البيانات.....9

2-2-2 تابع التجميع combiner function 13

2-3 نظام ملفات Hadoop الموزع HDFS.....15

2-3-1 تعريف HDFS (Hadoop Distributed File System) 15

2-3-2 تصميم HDFS 15

2-3-3 مفاهيم HDFS 16

2-3-4 التوافرية العالية (HDFS High Availability) 20

2-3-5 واجهات HDFS 20

2-3-6 تدفق البيانات في HDFS.....21

2-4 YARN 28

2-4-1 الشكل التفصيلي لعمل تطبيق YARN 29

2-4-2 دورة حياة التطبيق 31

2-4-3 مقارنة YARN بـ 1 Map-Reduce 32
2-4-4 الجدولة في YARN (Scheduling in YARN) 35
2-5 آلية عمل map reduce 37
2-5-1 آلية تنفيذ عمل map reduce 37
2-5-2 تسليم العمل 38
2-5-3 تهيئة العمل 39
2-5-4 تسليم المهمة 40
2-5-5 تنفيذ المهمة 41
2-5-6 التدفق 42
2-5-7 التقدم وتعديل الحالة 43
2-5-8 إتمام العمل 45
2-5-9 الفشل 46
2-5-9-1 فشل المهمة 46
2-5-9-2 فشل التطبيق الرئيسي 48
2-5-9-3 فشل مدير العقدة 49
2-5-9-4 فشل مدير الموارد 50
2-5-10 الخلط والفرز (Shuffle and Sort) 51
2-5-11 جلب خرج المقابلات. 54
2-5-12 ضبط الإعداد (Configuration tuning) 56
2-5-13 تنفيذ المهام 58
2-6 الخاتمة 62
الفصل الثالث – الدراسة المرجعية 63
3-1 المقدمة 63
3-2 محدودية محاكيات الحوسبة الشبكية 63
3-3 محاكيات المقابلة والاختزال 65
3-4 الخاتمة 77
الفصل الرابع – تصميم المحاكي HSG وتحقيقه 79

4-1 المقدمة 79
4-2 المتطلبات الواجب توافرها في المحاكي 79
4-3 مراحل تحقيق المتطلبات 79
4-3-1 اختيار المحاكي الشبكي الذي سيشكل البنية التحتية للمحاكي HSG 80
4-3-2 تصميم المحاكي HSG 89
4-3-3 تحقيق المحاكي 92
4-3-3-2 نظام الملفات الموزع (HDFS) 93
4-3-3-3 تحقيق YARN 97
4-3-3-4 تحقيق المقابلة والاختزال (Map Reduce) 104
4-4 الخاتمة 121
الفصل الخامس – الاختبارات العملية 123
5-1 المقدمة 123
5-2 اختبار نظام ملفات Hadoop الموزع (HDFS) 123
5-3 اختبار المحاكي 129
5-3-1 التحقق من دقة خرج المحاكي 129
5-3-2 متطلبات الذاكرة وأزمنة التنفيذ للمحاكي 136
5-3-3 تأثير أنماط الجدولة على زمن انهاء الاعمال 141
5-3-4 مقارنة HSG مع باقي المحاكيات 145
5-4 الخاتمة 148
الفصل السادس – الخاتمة والآفاق المستقبلية 150
6-1 المقدمة 150
6-2 ملخص الأطروحة 150
6-3 المساهمة التي قدمتها هذه الاطروحة 153
6-4 ملخص النتائج التي تم الحصول عليها 154
6-5 المقترحات المستقبلية 155
الفصل السابع – الملاحق 157
الملحق (1) 157

1- محاكاة الأحداث المتقطعة 157
2 الأنظمة والنماذج والمحاكاة (Systems ,models and simulation) 158
3 التجربة مع النظام الواقعي مقابل التجربة مع نموذج النظام 159
4 النموذج الفيزيائي مقابل النموذج الرياضي (Physical Model vs. Mathematical Model)..... 159
5 الطرائق التحليلية مقابل المحاكاة (Analytical solutions vs. Simulation) 159
6 أنواع النماذج 160
7 محاكاة الأحداث المتقطعة (Discrete event simulation)..... 161
7-1 طرائق تقديم الوقت (Time-Advance Mechanisms)..... 162
7-2 مكونات نموذج الأحداث المتقطعة وتنظيمها 164
7-3 محاكاة رتل بمخدم واحد 167
الملحق(2) 177
مثال عن المقابلة و الاختزال (بيانات الطقس) [2] 177
1 صيغة بيانات الطقس 177
2 معالجة البيانات باستخدام Hadoop..... 179
2-1 المقابلة والاختزال..... 179
2-2 المقابلة والاختزال في جافا 181
2-3 التنفيذ التجريبي 186
2-4 التوسع 192
2-5 تحديد تابع التجميع 192
المراجع 194

قائمة الأشكال والرسومات البيانية

الرقم	الشكل	الصفحة
1-2	C, تموضع البيانات ضمن الرف نفسه b ,تموضع البيانات على نفس العقدة a تموضع البيانات على رف مختلف	11
2-2	تدفق البيانات في إطار المقابلة والاختزال بوجود مهمة اختزال واحدة	12
3-2	تدفق البيانات في إطار المقابلة والاختزال بوجود أكثر من مهمة اختزال	13
4-2	تدفق البيانات في إطار Hadoop في حال عدم وجود مهام اختزال	14
5-2	قراءة الزبون من HDFS	21
6-2	المسافات الشبكية في Hadoop	24
7-2	كتابة البيانات على HDFS	25
8-2	الشكل النموذجي للخط الأنبوبي للتكرار	28
9-2	تطبيقات YARN	29
10-2	كيف يقوم YARN بتشغيل التطبيقات	30
11-2	أنواع الجدولة، FIFO I,CAPACITY II,FAIR III	36
12-2	كيف يقوم Hadoop بتشغيل عمل المقابلة والاختزال	38
13-2	علاقة التنفيذ التدفقي ومدير العقدة وحاوية المهمة	42
14-2	كيف تتم عملية تحديث وتوليد الحالة عبر نظام المقابلة والاختزال	45
15-2	الترتيب والخلط في المقابلة والاختزال	51
16-2	الدمج الفعال لـ 40 ملف بعامل دمج 10	55
1-3	أداء تطبيق TeraSort على المحاكي MRPerf	66
2-3	مقارنة أداء المحاكي MR مع التجربة الواقعية ومن أجل عناقيد غير متجانسة	73
1-4	مثال عن النموذج على مستوى التأخير	81
2-4	مثال عن النموذج من نوع Fluid	82
3-4	علاقة عدد العقد بسرعة المحاكاة عند محاكاة بروتوكول Chord وذلك بالنسبة لعدة محاكيات	84

4-4	البنية العامة للمحاكي HSG	89
5-4	المخطط التدفقي لتسلسل تنفيذ عمليات القراءة والكتابة على القرص الصلب	93
6-4	مخطط UML تسلسلي يبين آلية الكتابة على HDFS من أجل معامل تكرار 2	94
7-4	مخطط UML تسلسلي يمثل عملية القراءة من نظام الملفات الموزع.	96
8-4	آلية عمل مدير الموارد	98
9-4	المجدول FIFO	100
10-4	المجدول Fair	101
11-4	المجدول Capacity	102
12-4	مخطط UML يمثل أنماط الجدولة التي ترث الصف YarnSchedulerBase	104
13-4	مخطط UML تسلسلي يوضح آلية تنفيذ أعمال المقابلة والاختزال	105
14-4	مخطط تدفقي يوضح آلية تنفيذ مهمة المقابلة	110
15-4	آلية عمل الدامج	110
16-4	آلية عمل المختزل	115
17-4	ملف json يمثل عمل مقابلة واختزال	117
18-4	الملف cluster.json والذي يحدد خصائص العنقود الحاسوبي	119
19-4	نموذج عن ملف إعداد المجدول السعوي	119
20-4	مثال عن ملف نتائج أحد أعمال المقابلة والاختزال.	120
21-4	مثال عن احصائيات الارتال	120
1-5	اعدادات العقدة في HSG	124
2-5	أداء محاكاة عملية الكتابة على HDFS من أجل BS64و BS16	125
3-5	أداء عملية الكتابة على HDFS من أجل BS16,BS64	125
4-5	أداء محاكاة عملية الكتابة على HDFS من أجل BS128 وBS96	126
5-5	أداء عملية الكتابة على HDFS من أجل BS96,BS128	126
6-5	أداء محاكاة عملية القراءة من HDFS من أجل BS64و BS16	127

5-7	أداء عملية القراءة من HDFS من أجل BS16,BS64	127
5-8	أداء محاكاة عملية القراءة من HDFS من أجل BS128 وBS96	127
5-9	أداء عملية القراءة من HDFS من أجل BS96,BS128	128
5-10	عدد سجلات التفريغ خلال مرحلة المقابلة	131
5-11	عدد سجلات التفريغ خلال مرحلة الاختزال	132
5-12	عدد البايتات المقروءة خلال مرحلة المقابلة	132
5-13	عدد البايتات المقروءة خلال مرحلة الاختزال	133
5-14	عدد البايتات المكتوبة خلال مرحلة المقابلة	133
5-15	عدد البايتات المكتوبة خلال مرحلة الاختزال	134
5-16	زمن إنهاء العمل	134
5-17	الزمن اللازم من أجل عملية ال shuffle	135
5-18	إعدادات العنقود من أجل قياس الاستهلاك الذاكري والزمني في حالة تغير عدد العقد	137
5-19	زمن عملية المحاكاة عند تغير عدد العقد من 50-4100	138
5-20	تأثير تغير العقد على نتائج عملية المحاكاة (زمن اجراء التجربة)	138
5-21	الذاكرة التي يستهلكها المحاكي عند تغير عدد العقد من 50-4100	139
5-22	الذاكرة التي يستهلكها المحاكي عند تغير حجم العمل من G1 إلى G2048	139
5-23	زمن إجراء عند تغير حجم العمل من G1 إلى G2048	140
5-24	زمن المحاكاة في حالة نمط الجدولة (FIFO)	141
5-25	زمن المحاكاة في حالة نمط الجدولة (FAIR)	142
5-26	زمن المحاكاة في حالة نمط الجدولة 1 (Capacity)	143
5-27	زمن المحاكاة في حالة نمط الجدولة 2 (Capacity)	144

قائمة الجداول

الرقم	الجدول	الصفحة
1-1	مقارنة المقابلة والاختزال بنظم ادارة قواعد البيانات العلائقية (RDBMS)	3
1-2	مقارنة بين مكونات YARN و Map-Reduce 1	33
2-2	الاعدادات الواجب ضبطها من جهة عملية المقابلة	57
3-2	الاعدادات الواجب ضبطها من جهة عملية الاختزال	58
4-2	اهم الخصائص التي يمكن لمهام المقابلة والاختزال الوصول اليها والخاصة ببيئة التنفيذ	59
5-2	الخاصيات التي تتعلق بالتنفيذ التخميني	61
1-3	المتحولات التي تم استخدامها في HSim	71
2-3	مقارنة بين أهم محاكيات المقابلة والاختزال	75
1-4	مقارنة بين المحاكيات	83
1-5	مواصفات العقد التي استخدمت في المرجع [43] من أجل الاختبارات الواقعية على HDFS	123
2-5	مواصفات العقدة التي تعمل كمدير موارد وعقدة أسماء	129
3-5	مواصفات العقد التي تعمل كعقد بيانات ومدراء عقد	129
4-5	إعدادات العقد في التجربة الواقعية	131
5-5	المواصفات الفنية للحاسب المستخدم في قياس الاستهلاك الذاكري والزمني للمحاك.	136
6-5	احصائيات الرتل (FIFO)	141
7-5	احصائيات الرتل (Fair)	142
8-5	احصائيات الرتل (capacity q1 40%)	143
9-5	احصائيات الرتل (capacity q2 60%)	143
10-5	احصائيات الرتل (capacity q1 10%)	144
11-5	احصائيات الرتل (capacity q2 90%)	144
12-5	مقارنة بين عدد من محاكيات المقابلة والاختزال مع المحاكي HSG	145

5-13	اعدادات العمل و العنقود المستخدمة في [26]	146
5-14	زمن انجاز التجربة بالنسبة لـ MRSG وMRSim و MRPerf	146
5-15	زمن انجاز التجربة بالنسبة لـ MRSG وHSG	146

قائمة الاختصارات

HDFS	Hadoop Distributed File System	نظام الملفات الموزع الخاص بـ Hadoop
RDBMS	Relational Database Management System	نظام إدارة قواعد المعطيات العلائقية
HPC	High Performance Computing	الحوسبة عالية الأداء
SAN	Storage Area Network	شبكة منطقة التخزين
SETI@home	Search For Extraterrestrial Intelligence@home	البحث عن ذكاء خارج الأرض في المنزل
YARN	Yet Another Resource Negotiator	بعد مفاوض موارد آخر
API	Application Programming Interface	واجهة التطبيقات البرمجية
RPC	Remote Procedure Call	استدعاء الطرائق عن بعد
FIFO	First In First Out	الدخل أولاً خارج أولاً
HA	High Availability	التوافرية العالية
HDD	Hard Disk Drive	محرك القرص الصلب
DES	Discrete Event Simulation	محاكاة الأحداث المتقطعة
PDES	Parallel Discrete Event Simulation	محاكاة الأحداث المتقطعة الموزعة
NCDC	National Climatic Data Center	المركز الوطني للبيانات المناخية
ASCII	American Standard Code for Information Interchange	الكود القياسي الأمريكي لتبادل المعلومات
JVM	Java Virtual Machine	آلة جافا الافتراضية
MR	Map Reduce	المقابلة والاختزال
ACID	Atomicity, Consistency, Isolation, Durability	العملية ذرية ومتسقة ومعزولة عن باقي العمليات وغير قابلة للتراجع

الفصل الأول

المقدمة والهدف من البحث

1-1 المقدمة

تم اقتراح إطار المقابلة والاختزال من قبل Google[1] في عام 2004 من أجل معالجة و تحليل البيانات الضخمة، وظهر فيما بعد العديد من التحقيقات لهذا الإطار، إلا أن أهمها وأكثرها انتشاراً هو Hadoop. بالتالي، سنعتمد على التحقيق Hadoop عند محاكاة إطار المقابلة والاختزال، وسنعتبر Hadoop مكافئاً لكلمة إطار المقابلة والاختزال في هذا البحث علما أن Hadoop مشروعاً مستقلاً يهدف لمعالجة البيانات الضخمة والمخزنة على نظام ملفات موزع خاص هو Hadoop) HDFS Distributed File System) ويحتوي على مدير موارد خاص به YARN (Yet Another Resource Negotiator).

ليس من السهل قياس كمية البيانات المخزنة إلكترونياً، لكن هنالك تقريراً لـ IDC يتوقع حجم البيانات في عام 2013 بحدود 4.4ZB[1]، ومن المرجح أن يصبح حجم البيانات بحدود 44ZB بحلول العام 2020 [2, p. 3]. إن هذه الكمية الضخمة من البيانات تأتي من العديد من المصادر أهمها:

1- سوق الأوراق المالية في نيويورك يولد 4-5 terabyte يومياً.
2- موقع facebook يولد 7petabytes شهريا.
3- موقع ancestry.com الخاص بعلم الوراثة، يولد 10betabytes يوميا.

إنه أمراً جيداً أن يكون لدينا مصادر وكميات كبيرة من البيانات إلا أن المشكلة تكمن في إمكانية تخزين وتحليل هذه البيانات.

1-1 تخزين البيانات وتحليلها

على الرغم من ازدياد قدرة الأقراص الصلبة على تخزين البيانات لكن سرعة القراءة والكتابة على هذه الأقراص لم تزداد بالمقدار نفسه. فعلى سبيل المثال، في عام 1995 كان من الممكن تخزين 1370MB على قرص صلب يتمتع بسرعة نقل تصل إلى4.4 MB/s وبالتالي يمكنه قراءة كل ما

[1] (Zetta byte =10^{21}byte)

في القرص خلال خمس دقائق. ولكن وبعد عشرين عاماً أصبح بالإمكان تخزين 1TB على قرص صلب واحد بسرعة نقل لا تتجاوز 100 MB/s، وبالتالي سنحتاج إلى أكثر من ساعتين ونصف لقراءة كل ما في القرص الصلب.

إن هذا الوقت يعتبر وقتا كبيرا لقراءة قرص صلب واحد فكيف الحال مع الكتابة التي ستستغرق وقتا أطول. إن أفضل طريقة لتقليل الوقت هي القراءة من أكثر من قرص في نفس الوقت. ولنفترض أننا نملك 100 قرص وكل قرص يتسع لـ 1 Terabyte كحد أقصى، فإذا قمنا بتخزين 1 بالمئة من البيانات المخزنة على قرص واحد (والتي كان حجمها بحدود 1 Terabyte) بالتالي سيكون الوقت اللازم لقراءة هذه البيانات مساويا لدقيقتين تقريبا. إن استخدام 1% من القدرة التخزينية للأقراص هو هدر في القدرة التخزينية، لكن يمكننا أن نخزن 100 مجموعة من البيانات بحيث يكون حجم كل مجموعة بحدود 1 TB [2, p. 3]

إن هذا الحل يساعد في زيادة سرعة القراءة والكتابة إلا أنه يشتمل على العديد من المشاكل وأهم هذه المشاكل:

1- المشكلة الأولى هي فشل العتاد الصلب: بما أننا نستخدم العديد من قطع العتاد الصلب بالتالي إمكانية الخطأ أصبحت أكبر وبالتالي خسارة البيانات أصبحت أمرا متوقع الحدوث. أمام هذا التحدي لا بد من وجود ألية للنسخ الاحتياطي والتكرار كما هو الحال باستخدام تقنية RAID أو باستخدام نظام ملفات موزع مثل Hadoop Distributed File System (HDFS) الخاص بإطار Hadoop.
2- المشكلة الثانية هي أن معظم مهام التحليل تحتاج لدمج البيانات وهذه البيانات يمكن أن تتواجد على أقراص مختلفة. إن هذه العملية هي عملية ليست بهذه البساطة وهنالك العديد من النظم الموزعة التي تساعد على ذلك لعل أهمها نموذج المقابلة والاختزال الذي سنتحدث عنه [2, p. 5].

1-2 مجال عمل Hadoop

إن قوة Hadoop تأتي من مناسبته للأعمال الدفعية، وعدم ملائمته للتحليل التفاعلي. فمن غير الممكن الاستعلام والحصول على نتائج ببضع ثواني أو أقل. بشكل عام الاستعلامات في Hadoop تستهلك عدة دقائق أو أكثر. أي أن Hadoop مناسب للاستخدام من نوع Offline بحيث لا يتدخل

المستخدم بعملية المعالجة أثناء انتظار النتائج. لكن Hadoop فيما بعد أصبح يشتمل على أمور أبعد من التنفيذ الدفعي. لان Hadoop أصبح اسم يدل على عدد من المشاريع (ليس فقط HDFS and Map–Reduce).

1–3 مقارنة Hadoop مع باقي الأنظمة

1–3–1 أنظمة إدارة قواعد البيانات العلائقية

يستخدم Hadoop مع عدد كبير من الأقراص من أجل تحليل البيانات واسع النطاق ولا يمكن استخدام قواعد المعطيات العلائقية بنفس الكفاءة في هذه الحالة.

يتعلق هذا الأمر بالقرص الصلب وزمن الطلب (seek time) له وهو الزمن اللازم لوصول رأس القراءة والكتابة إلى المكان المطلوب. ويعبر هذا الزمن عن التأخر في تنفيذ عملية القراءة أو الكتابة وهذا الزمن ليس بقليل. ولنفترض ولتعديل جزء من البيانات في قاعدة البيانات وللمرور على شجرة البحث الثنائي سيقوم رأس القراءة والكتابة بالانتقال عدة مرات وكل انتقال سيكلف فترة زمنية من رتبة الميلي ثانية. ولتعديل معظم مجموعة البيانات سيكون أسلوب البحث الثنائي أقل فاعلية من نموذج المقابلة والاختزال الذي يستخدم أسلوب الترتيب والدمج لإعادة بناء قاعدة البيانات وفي الجدول التالي أهم الفروقات بين نظام إدارة قواعد المعطيات وHadoop [2, p. 8].

المعيار	قواعد المعطيات العلائقية	المقابلة والاختزال
حجم البيانات	Gigabytes	Petabytes
الوصول	تفاعلي ودفعي	دفعي
معدل التعديل	تتم القراءة والكتابة عدة مرات	تتم الكتابة مرة واحدة والقراءة مرات عدة
العمليات	ACID	لا يوجد
الهرمية	Schema on write	Schema on read
التكامل	عالي	منخفض
التوسع	غير خطي	خطي

الجدول (1–1) مقارنة المقابلة والاختزال بنظم ادارة قواعد البيانات العلائقية [2, p. 8].

1-3-2 الحوسبة الشبكية

إن مجموعات الحوسبة عالية الأداء (HPC high performance computing) والحوسبة الشبكية (Grid computing) قامت ولسنين عديدة بمعالجة البيانات واسعة النطاق، إن هذه التطبيقات تستخدم تقنية تمرير الرسائل[1]. بشكل عام إن تقنية HPC تعتمد على توزيع العمل على مجموعة من الطرفيات ضمن العنقود، هذه الطرفيات يمكنها أن تصل إلى نظام ملفات مشترك، يتم استضافة هذا النظام من قبل شبكة تخزين (Storage Area Network SAN). إن هذه الطريقة تعمل بشكل جيد من أجل الأعمال التي تتطلب طاقة حسابية كبيرة، ولكن المشاكل تبدأ بالظهور عند الحاجة للوصول إلى كمية كبيرة من البيانات (من رتبة المئات من الغيغابايت)، حيث يلمع نجم Hadoop ابتداء من هذه النقطة، وذلك لأن عرض حزمة الشبكة يشكل عنق زجاجة ولأن العقد الحسابية تصبح عاطلة عن العمل.

يحاول Hadoop قدر الإمكان أن يجعل البيانات محلية بالنسبة للعقد الحسابية، وبالتالي سيصبح الوصول إلى البيانات أسرع. وتعرف هذه الخاصية بتموضع البيانات محليا (data locality)، وتعتبر بمثابة القلب بالنسبة لمعالجة البيانات في Hadoop كما أنها السبب الذي يمنح Hadoop أدائه العالي.

إن أسلوب تمرير الرسائل يمنح المبرمجين قدرات تحكم كبيرة، ولكنه يتطلب آليات صريحة للتحكم بتدفق البيانات (مثل دوال لغة C وإعداد المقابس Sockets). إن معالجة البيانات في Hadoop تعمل بمستوى أعلى: لأن المبرمج يفكر فقط بنموذج البيانات (مثل أزواج قيمة–مفتاح في المقابلة والاختزال)، بينما يكون تدفق البيانات ضمنيا [3].

1-3-3 الحوسبة التطوعية

إن مشاريع الحوسبة التطوعية تعمل من خلال تقسيم المشكلة المراد حلها إلى وحدات عمل صغيرة والتي ترسل إلى حواسيب المتطوعين عبر العالم، على سبيل المثال SETI@home يقسم المسألة إلى وحدات عمل بحجم 0.35 ميغا بايت من البيانات، والتي تستهلك ساعات وربما أيام من التحليل على الحواسيب الشخصية. عندما يتم الانتهاء من عملية تحليل وحدة العمل يتم إرسال النتيجة إلى

[1] يشير تمرير الرسائل، إلى إرسال رسالة إلى عملية يمكن أن تكون كائنًا أو دالة، ويمكن استخدام هذه الرسالة لاستدعاء عملية أخرى بشكل مباشر أو غير مباشر. يعد تمرير الرسائل مفيدًا بشكل خاص في البرمجة غرضية التوجه والبرمجة التفرعية عندما يتم إرسال رسالة واحدة (في شكل إشارة أو حزمة بيانات) إلى المستلم.

المخدم ليستلم الجهاز المتطوع مهمة أخرى. ومن أجل ضمان عدم إرسال نتائج خاطئة يتم إرسال نفس وحدة العمل إلى ثلاثة متطوعين وإذا تطابقت نتيجتان تعتبر النتيجة صحية ويتم اعتمادها[4].

هنالك تشابه بين Hadoop والحوسبة التطوعية من حيث تقسيم المشكلة الكبيرة إلى أجزاء لكن هنالك فروقات جوهرية أهمها:

1- إن مهام الحوسبة التطوعية مستهلك للمعالجة بشكل كبير وتستهلك وقتا كبيرا وحجم العمل المرسل صغير بالتالي موضوع عرض الحزمة ليس بعامل مؤثر.
2- تم تصميم Hadoop من أجل الأعمال التي تستهلك دقائق أو ساعات أو أكثر من ذلك ولكن يجب أن تكون الشبكة التي تربط العقد عالية السرعة نظرا لحجوم البيانات الممكن تبادلها وخصوصا في مرحلة الاختزال[1].

إن الحوسبة التطوعية غير ملائمة لأعمال معالجة وتحليل البيانات الضخمة نظرا للفروقات السابقة حيث ترتبط حواسيب المتطوعين عبر شبكة الانترنت حيث تكون السرعة محدودة بشكل عام وبالتالي كمية البيانات التي يمكن تبادلها محدود.

1-4 دوافع البحث

بعد أن استعرضنا في المقدمة السابقة أهمية Hadoop في معالجة البيانات الضخمة وتمايزه عن باقي البنى الموزعة. سنبين فيما يأتي الأسباب التي تدفعنا لبناء محاك لهذا الإطار كما سنتكلم عن الأهداف المنشودة من هذا البحث.

إن عملية معالجة البيانات الضخمة تختلف باختلاف المهمة المراد تنفيذها والبيانات المراد معالجتها. **والمهام** تختلف عن بعضها من حيث:

1- استهلاكها للمعالج
2- حجم معطيات القراءة والكتابة على القرص الصلب
3- حجم معطيات القراءة والكتابة على نظام الملفات الموزع
4- حجم المعطيات المتبادلة على الشبكة
5- متطلبات الذاكرة

أما **البيانات** فتختلف عن بعضها من حيث:

1- الحجم
2- التوزع على العقد والأقراص
3- عدد النسخ
4- الصيغ المخزنة
5- الضغط والتشفير

إن هذه الاختلافات وغيرها تؤثر في زمن تنفيذ المهام وفي استهلاك الموارد، وبالتالي أي تعديل بسيط في إعدادات المهام والعناقيد الحاسوبية يصبح له أثر واضح على الأداء وعلى استهلاك الموارد ويصبح الوصول إلى أفضل أداء أمرا صعبا إن لم يكن مستحيلا. فبعض الإعدادات تناسب حالات معينة ومهام معينة وقد لا تتناسب مع حالات أخرى. هنا ندرك اهمية وجود محاك لإطار المقابلة والاختزال يساعد في ضبط أعمال المقابلة والاختزال قبل إجراء العمليات الفعلية على العناقيد. ويساعد الباحثين في مجال الجدولة من تجريب ودراسة خوارزميات جدولة مخصصة وتأثيرها على استخدامية العناقيد الحاسوبية واستهلاك الموارد.

1-5 الأهداف

1. دراسة إطار المقابلة والاختزال و محاكياته
2. تطوير محاك جديد لإطار المقابلة والاختزال يتلافى محدوديات المحاكيات الأخرى من حيث:
   - التوسعية (القدرة على محاكات عناقيد حاسوبية تحتوي على آلاف العقد)
   - قابلية التوسعة (القدرة على إضافة ميزات جديدة للمحاكي)
   - نمذجة نظام الملفات الموزع HDFS
   - سهولة الاستخدام (إعداد التجارب واصدار النتائج)
   - تحمل الفشل
   - القدرة على محاكات شبكات متجانسة وغير متجانسة
   - دعم أكثر من قرص صلب على نفس الجهاز

1-6 المساهمة

1- إعداد دراسة مرجعية تفصيلية لمحاكيات المقابلة والاختزال وقد ختمنا هذه الدراسة بجدول يلخص خصائص كل محاك.

2- تقديم المحاكي (Hadoop SimGrid) HSGوهو محاكي لإطار المقابلة والاختزال محقق بلغة ++Cوبالاعتماد على المحاكي SimGrid حيث:

- دعم عمليات القراءة والكتابة والتكرار على HDFSبشكل مفصل وهذا ما لم يتم لحظه بهذا التفصيل في باقي المحاكيات كما تم تحقيق مدير الموارد بشكل مفصل وذلك بسبب اعتماد تطبيق المقابلة والاختزال على هذان المكونان.
- كما أن الفصل بين مكونات المحاكي يمكن من استخدام اجزاء منه مثل HDFS وYARN لدراسة أطر أخرى مثل Hbase و Spark.
- يمكن إضافة مجدولات جديدة على المحاكي دون الحاجة للتعديل في بنيته الأساسية.
- إن المحاكي HSGأعطى سرعة أفضل بكثير مقارنة بالمحاكي MRPerf و المحاكي MRSim و تمت المقارنة مع هذان المحاكيات نظرا لأنهما يراعيان تفاصيل العقد بشكل مشابه ل HSG.
- إن المحاكي SimGrid يستخدم ملفات XML تعبر عن هيكلية الشبكة وتوزع العقد ضمن الشبكة واللذان يصعب اعدادهما بالتالي قمنا بدعم المحاكي HSG بالقدرة على توليد هذه الملفات باستخدام ملف اعدادات JSON مما يسهل اعداد التجارب.

1-7 اقسام الرسالة

بعد أن استعرضنا في الفصل الأول مقدمة تمهد للبحث بالإضافة إلى الدوافع والأهداف والمساهمة المحققة، سنتحدث في الفصل الثاني عن إطار المقابلة والاختزال وآلية عمله، بعدها سنستعرض أهم الجهود المبذولة في مجال محاكاة المقابلة والاختزال وذلك في الفصل الثالث، سننتقل في الفصل الرابع إلى المحاكي HSG الذي قمنا بتصميمه وتحقيقه مبينين تفاصيل كل مكون من مكونات هذا المحاكي، و سيحتوي الفصل الخامس على الاختبارات التي تمت على المحاكي HSG من أجل التحقق من دقة نتائجه و استهلاكه الذاكري و الزمني و مقارنته ببعض المحاكيات، بعدها تأتي الخاتمة والآفاق المستقبلية في الفصل السادس. أما المراجع والملاحق فستكون في نهاية الرسالة.

الفصل الثاني

المقابلة والاختزال

2–1 المقدمة

يقدم لنا هذا الفصل فهم عميق لإطار المقابلة والاختزال Map–Reduce بالإضافة إلى نظام الملفات الموزع ومدير الموارد في Hadoop، إذ اعتمدنا على هذه التفاصيل من أجل تحقيق المحاكي HSG والذي سنتكلم عنه في الفصل الرابع.

2–2 آلية عمل إطار المقابلة والاختزال

يقوم إطار المقابلة والاختزال بتقسيم معالجة البيانات إلى مرحلتين:

1– مرحلة المقابلة Map phase.

2– مرحلة الاختزال Reduce phase.

وكل مرحلة تحتوي على دخل وخرج بشكل مفتاح–قيمة. يتم اختيار نوع المفاتيح أو القيم من قبل المبرمج. وعلى المبرمج أن يحدد دالتين هما: دالة المقابلة ودالة الاختزال.

قمنا بتوضيح أسلوب عمل إطار المقابلة والاختزال من خلال مثال عملي يستخدم بيانات الطقس وذلك في الملحق (2) في الصفحة 194.

2–2–1 تدفق البيانات

تعاريف:

1– عمل المقابلة والاختزال Map–Reduce job: هو عبارة عن وحدة من العمل التي يريد المستخدم تنفيذها، وهو يتألف من بيانات الدخل وبرنامج المقابلة والاختزال ومعلومات الإعداد.

2– المهام Tasks: يقوم Hadoop بتنفيذ العمل من خلال تقسيمه إلى مهام وهنالك نوعين من المهام هي: مهام المقابلة ومهام الاختزال. تتم جدولة المهام باستخدام YARN ويتم تنفيذها على العقد ضمن العنقود الحاسوبي. وفي حال فشل المهمة، يقوم المجدول بإعادة جدولتها على عقدة مختلفة.

يقوم Hadoop بتقسيم بيانات الدخل من أجل عمل المقابلة والاختزال، وتدعى هذه الأقسام بأقسام الدخل input splits أو أنها تدعى أقسام فقط وتكون هذه الأقسام متساوية الحجم. يقوم Hadoop بإنشاء مهمة مقابلة من أجل كل قسم، هذه المهمة تقوم بتنفيذ تابع المقابلة المعرف من قبل المستخدم والذي يقوم بمعالجة كل سجلات قسم الدخل.

في حال كان لدينا العديد من الأقسام فإن الوقت اللازم لمعالجة كل قسم هو وقت قليل مقارنة بتنفيذ كامل العمل. بالتالي في حال تم تنفيذ هذه الأقسام على التفرع سيصبح التنفيذ أفضل من ناحية توزيع الحمل، علما أن العقد الأسرع ستقوم بتنفيذ أعمال أكثر من العقد ذات السرعات الأدنى. وحتى لو كانت العقد متماثلة، فإن فشل العمليات أو وجود أعمال يتم تنفيذها على التوازي سيجعل موضوع موازنة الأحمال أمرا ضروريا وإن جودة موازنة الأحمال تزداد عندما تكون عملية التقسيم جيدة.

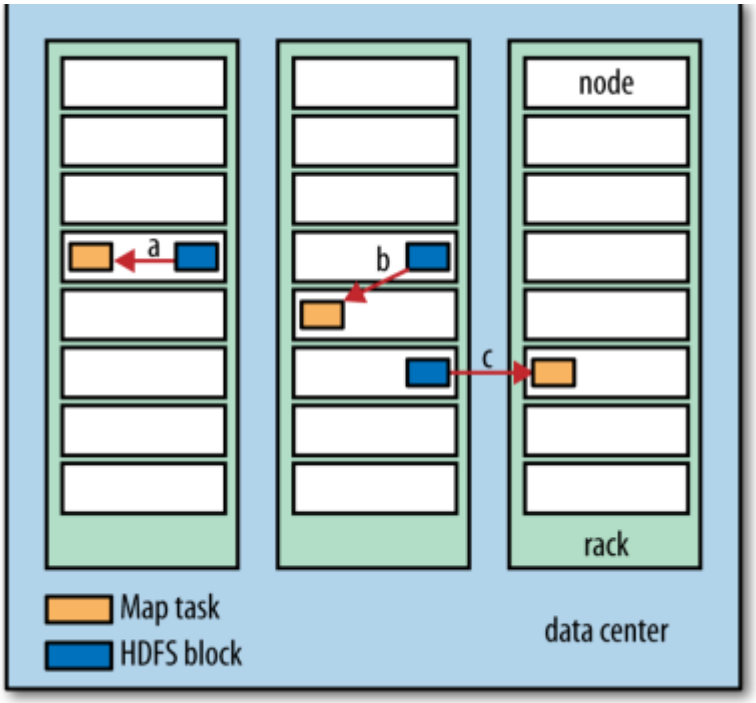
وفي المقابل، وفي حال كانت الأعمال صغيرة جدا، ستكون كلفة إدارة الأقسام وإنشاء مهام المقابلة أمرا مؤثر على زمن تنفيذ العمل ككل. إن حجم القسم من أجل معظم الأعمال هو نفسه حجم كتلة نظام الملفات الموزع HDFS block(128MB بشكل افتراضي)، علما أنه يمكن تغير هذه القيمة في العنقود الحاسوبي (من أجل كل الملفات التي سيتم إنشائها حديثا) أو من أجل أي ملف سيتم إنشاؤه [2,p.31].

إن Hadoop يسعى لتنفيذ كل مهمة مقابلة على العقدة التي تحتوي على بيانات الدخل الضرورية للمهمة، وذلك لأن Hadoop لا يستخدم عناقيد حاسوبية ذات سرعات شبكية عالية. إن هذه الخاصية تدعى أمثلة تموضع البيانات محليا data locality optimization. في بعض الأحيان تكون كل العقد التي تحتوي نسخ من البيانات مشغولة (تقوم بتنفيذ مهام أخرى)، بالتالي سيقوم المجدول بالبحث عن مكان فارغ لعملية المقابلة ضمن عقدة تنتمي لنفس الرف الذي يحتوي على البيانات. ممكن وفي بعض الحالات أن يتعذر إيجاد مكان فارغ لعملية المقابلة ضمن نفس الرف، بالتالي سيتم البحث عن مكان فارغ لعملية المقابلة ضمن رف آخر وهذا يدعى off–rack node والشكل (2–1) يبين الحالات الثلاث.

أصبح من الواضح الآن السبب الذي يجعل الحجم الأمثلي لقسم بيانات الدخل في عملية المقابلة مساويا لحجم كتلة HDFS: لأن هذا الحجم هو أكبر حجم لبيانات الدخل والتي يمكن أن تكون

مخزنة على عقدة واحدة. في حال كان القسم يشتمل على أكثر من كتلة، سيتوجب نقل كتلة عبر الشبكة لأنه على الأغلب لن يتواجد كتلتين على نفس العقدة، عملية النقل ستكون من العقدة التي تحتوي البيانات إلى العقدة التي تقوم بتشغيل مهمة المقابلة.

إن مهام المقابلة تقوم بكتابة خرجها إلى القرص المحلي، وليس إلى HDFS وذلك لأن خرج عملية المقابلة هو خرج وسيطي: حيث تتم معالجته من قبل مهام الاختزال وذلك للوصول إلى النتيجة النهائية، وحالما يتم الانتهاء من العمل، يمكن التخلص من خرج مهام المقابلة. بالتالي فإن تخزينه على نظام الملفات الموزع مع وجود عملية التكرار للبيانات (replication) سيؤثر بشكل سلبي على الأداء ودون أي فائدة. في حال فشل عقدة تقوم بعملية مقابلة وكان الفشل قبل أن تقوم مهمة الاختزال باستهلاك خرج المقابلة، سيقوم Hadoop بشكل آلي بإعادة تنفيذ عملية المقابلة على عقدة أخرى من أجل إعادة إنشاء خرج المقابلة المفقود.

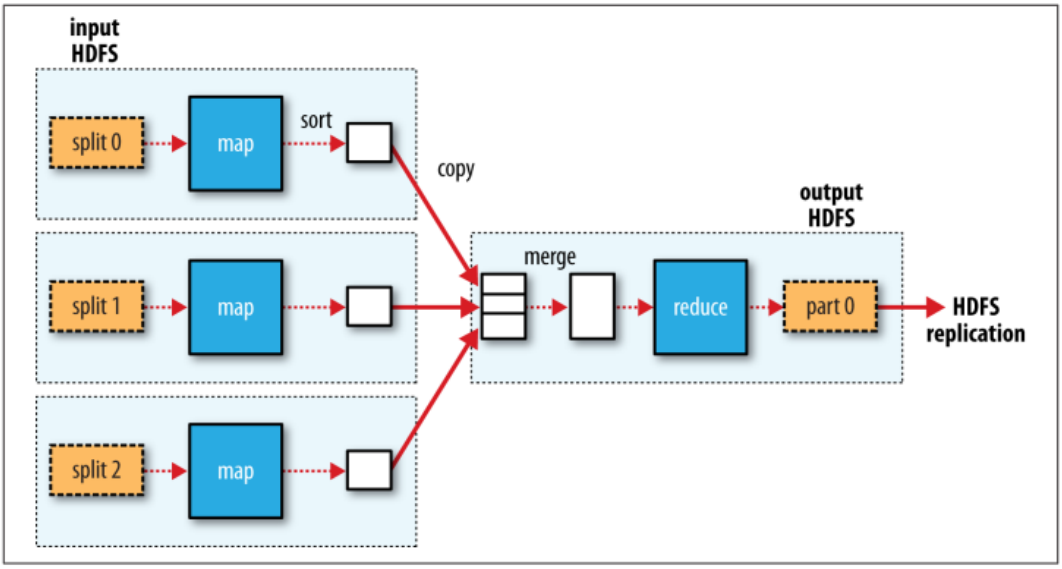


الشكل (2–1) (a) تموضع البيانات على نفس عقدة التنفيذ، (b) عقدة البيانات و عقدة التنفيذ ضمن الرف نفسه، (C)عقدة البيانات وعقدة التنفيذ على رفوف مختلفة[2, p. 31]

إن عمليات الاختزال لا تستفيد من حسنات تموضع البيانات محلياً: لأنه وبشكل طبيعي سيكون دخل عملية الاختزال هو خرج كل عمليات المقابلة. في مثال بيانات الطقس (الملحق 2) كان لدينا عملية اختزال وحيدة تقوم بأخذ خرج عمليات المقابلة جميعا. بالتالي لا بد من نقل خرج علميات المقابلة

بصيغة مرتبة عبر الشبكة ليصل إلى مهام الاختزال التي يتم تنفيذها، حيث تقوم مهام الاختزال بدمج خرج المقابلة وتمرره إلى تابع الاختزال المعرف من قبل المستخدم. عادة ما يتم تخزين خرج عملية الاختزال ضمن HDFS بحيث يتم تكراره. كما سنرى في هذا الفصل، ومن أجل كل كتلة من HDFS تحتوي على خرج عملية الاختزال ستتم عملية التكرار بالشكل التالي: أول نسخة يتم تخزينها على العقدة محليا، باقي النسخ يتم تخزينها على عقد لا تنتمي إلى نفس الرف.

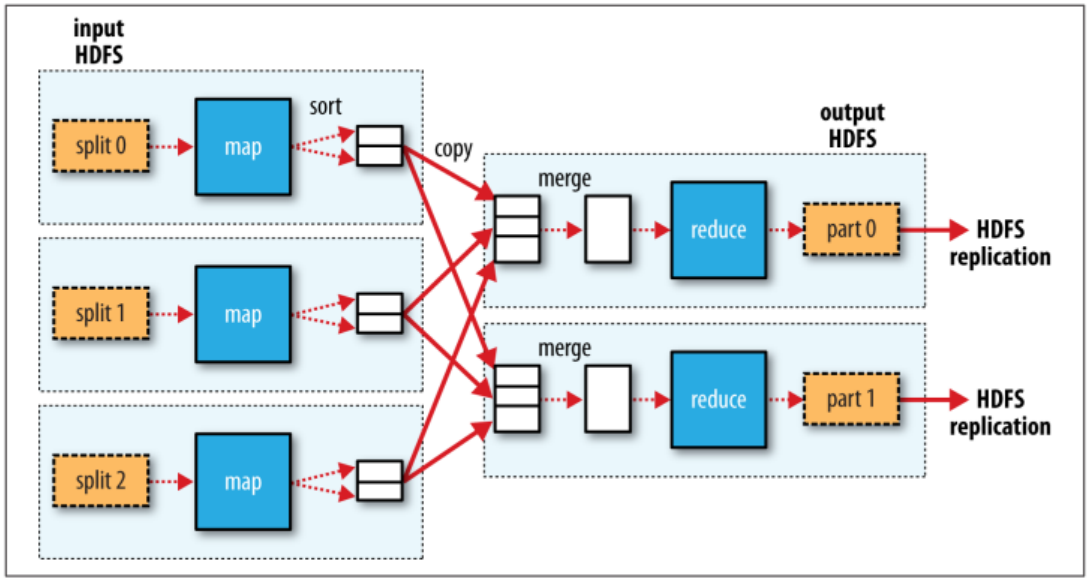
الشكل (2–2) يبين التدفق الكامل للبيانات لعملية المقابلة والاختزال وذلك من أجل مهمة اختزال واحدة. الصناديق المنقطة تعبر عن العقد، أما الأسهم المنقطة فهي تعبر عن نقل البيانات ضمن العقدة، أما الأسهم الغير منقطة فهي تعبر عن تدفق البيانات بين العقد.



الشكل (2–2) تدفق البيانات في إطار المقابلة والاختزال بوجود مهمة اختزال واحدة[2, p. 33].

إن عدد مهام الاختزال لا يتعلق بحجم بيانات الدخل، وإنما يتم تحديده بشكل منفصل. عندما يكون لدينا أكثر من مختزل، تقوم مهام المقابلة بتقسيم خرجها بحيث يكون لكل مهمة مقابلة خرج منفصل من أجل كل مهمة اختزال. يمكن أن يكون لدينا أكثر من مفتاح في كل قسم (مفتاح مع القيمة الخاصة به)، ولكن السجلات من أجل أي مفتاح يجب أن تكون في نفس القسم. إن عملية التقسيم يمكن أن يتم التحكم بها من قبل تابع تقسيم تتم كتابته من قبل المستخدم، لكن عادة ما يقوم المقسم الافتراضي بعملية التقسيم بشكل جيد وذلك من خلال استخدام تابع تقطيع (hash function).

إن تدفق البيانات في إطار المقابلة والاختزال وعند وجود أكثر من مهمة اختزال مبين بالشكل (2–3) إن هذا الشكل يبين بشكل واضح سبب تسمية تدفق البينات بين عمليات المقابلة والاختزال بالخلط (shuffle)، وذلك لأن كل مهمة اختزال يتم تغذيتها من قبل عدة مهام مقابلة. إن عملية الخلط معقدة بشكل أكبر مما هي عليه في الشكل(2–3). كما أن ضبط عملية الخلط يلعب دور كبير في زمن تنفيذ العمل.



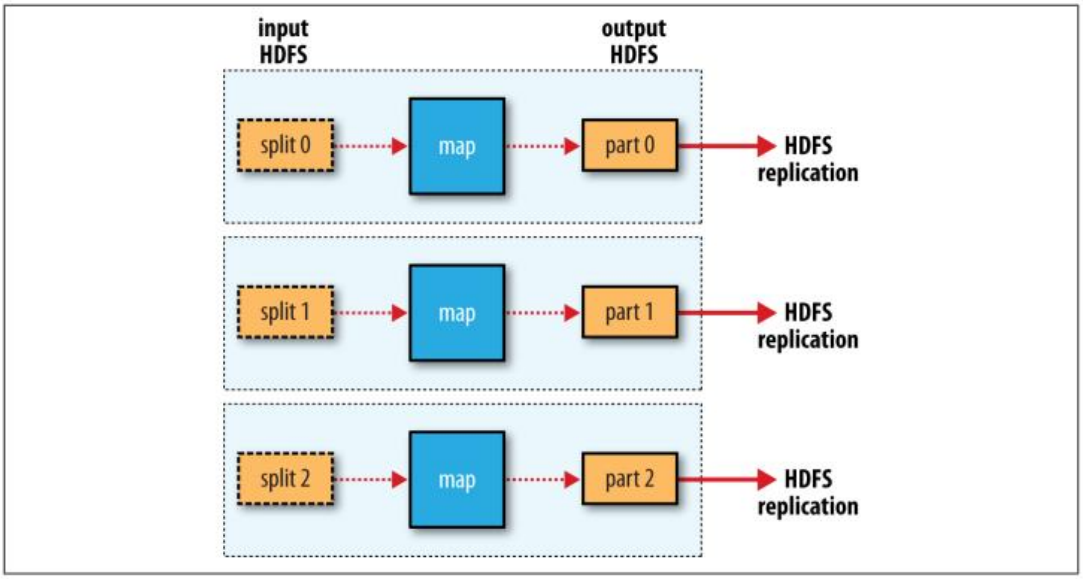
الشكل (2–3) تدفق البيانات في إطار المقابلة والاختزال بوجود أكثر من مهمة اختزال [2, p. 34]

كما يمكن تنفيذ بعض الأعمال دون الحاجة إلى مهام اختزال، بالتالي لا يوجد حاجة لعملية الخلط لأن معالجة البيانات في هذه الحالة يمكن تنفيذها بشكل موزع بشكل كامل.

2–2–2 تابع التجميع combiner function

إن العديد من أعمال المقابلة والاختزال محدودة بعرض الحزمة المتوفر في العنقود الحاسوبي.

إن Hadoop يسمح للمستخدم بتعريف تابع التجميع combiner function بحيث يتم تنفيذه على خرج المقابلة، كما أن خرج تابع التجميع يمثل دخل تابع الاختزال. وبما أن تابع التجميع يعتبر شكل من أشكال التحسين، فإن Hadoop لا يقدم ضمانة عن عدد المرات التي يتم فيها استدعاء هذا التابع من أجل كمية محددة من سجلات خرج الدخل. بعبارة أخرى فإن عدم استدعاء تابع التجميع أو استدعائه مرة واحدة أو عدة مرات يجب ألا يؤثر على خرج تابع الاختزال.



الشكل (2–4) تدفق البيانات في إطار Hadoop في حال عدم وجود مهام اختزال [2, p. 35]

يحدد تابع التجميع أنواع التوابع الممكن استخدامها لذلك. لتوضيح ذلك سنقدم المثال التالي: ليكن لدينا المثال الخاص بدرجة الحرارة الأعظمية (مثال بيانات الطقس و الموضح في الملحق (2))، وليكن لدينا درجات الحرارة الخاصة بالعام 1950 حيث تمت معالجتها على مقابلين (بسبب وجود قسمين من البيانات يحتويان درجات حرارة ذلك العام). وكان خرج أول مقابل:

```
(1950, 0)
(1950, 20)
(1950, 10)
```

وخرج ثاني مقابل:

```
(1950, 25)
(1950, 15)
```

وسيكون دخل تابع الاختزال بالشكل التالي:

```
(1950, [0, 20, 10, 25, 15])
```

والخرج النهائي:

```
(1950, 25)
```

وذلك لأن الـ 25 هي أكبر عدد في السلسلة. يمكننا أن نستخدم تابع التجميع، بشكل مشابه لتابع الاختزال، سيتم ايجاد درجة الحرارة العظمة من أجل كل مقابلة، وسيكون دخل تابع الاختزال بالشكل التالي:

```
(1950, [20, 25])
```

وسينتج نفس النتيجة، يمكننا شرح استدعاء التوابع على قيم درجات الحرارة بالشكل التالي:

```
max(0, 20, 10, 25, 15) = max(max(0, 20, 10), max(25, 15)) = max(20, 25) = 25
```

هذا النوع من التوابع يدعى توابع توزيعية. ولكن هنالك عدد من التوابع التي لا تندرج تحت هذا النوع مثل تابع الوسيط حيث:

```
mean(0, 20, 10, 25, 15) = 14
```

but:

```
mean(mean(0, 20, 10), mean(25, 15)) = mean(10, 20) = 15
```

إن تابع التجميع لا يلغي دور تابع الاختزال. (لأننا ما زلنا بحاجة تابع الاختزال من أجل معالجة سجلات تحتوي على نفس المفتاح ولكن تنتمي لمهام مقابلة مختلفة) لكن وكما ذكرنا سابقا فإن تابع الدمج يساعد على تقليل كمية البيانات التي سيتم خلطها بين المقابلات والمختزلات، وهذا السبب الذي يدفعنا لاستخدام تابع التجميع (طبعا عندما يكون هنالك امكانية لتقليل كمية البيانات)[2, p. 36].

2-3 نظام ملفات Hadoop الموزع HDFS

2-3-1 تعريف HDFS (Hadoop Distributed File System)

عندما تنمو مجموعة البيانات بحيث يصبح من الصعب تخزينها ضمن آلة فيزيائية واحدة، يصبح من الضروري تقسيم مجموعة البيانات من أجل تخزينها على عدة آلات(حواسيب). إن نظام الملفات الذي يدير عملية التخزين عبر الشبكة يدعى نظام ملفات موزع distributed file systems. وبما ن نظام الملفات الموزع يعتمد على الشبكة بما تحويه من أمور برمجية بالتالي سيكون نظام الملفات الموزع معقد بشكل أكبر من نظام الملفات الخاص بالقرص الصلب العادي. على سبيل المثال، إن أهم التحديات هي جعل نظام الملفات يتحمل فشل بعض العقد دون ضياع البيانات.

إن Hadoop يؤمن نظام ملفات موزع يدعى Hadoop Distributed FileSystem(HDFS) و هو نظام ملفات متعدد الأغراض ومجرد [2, p. 43].

2-3-2 تصميم HDFS

إن HDFS هو نظام ملفات تم تصميمه من أجل تخزين الملفات ذات الحجوم الضخمة مع إمكانية الوصول التدفقي للبيانات وكذلك التشغيل على عناقيد حاسوبية تحتوي على عتاد صلب متوفر ورخيص نسبيا. سنقدم بعض الشرح عن هذه الجملة[2, p. 44].

1– الملفات ذات الحجوم الضخمة: هذا يعني الملفات التي يصل حجمها إلى مئات من الميغابايتات أو الغيغابايتات أو التيرابايتات. في هذه الأيام لدينا عناقيد Hadoop تقوم بتخزين بيتا بايت من البيانات.

2– الوصول التدفقي للبيانات: تم بناء Hadoop بحيث يراعي فكرة مهمة في أنماط معالجة البيانات وهي (الكتابة لمرة واحدة – القراءة لعدة مرات). إن مجموعة البينات يتم توليدها أو نسخها من المصدر، تخضع بعدها للعديد من عمليات التحليل. وكل عملية تحليل يمكن أن تستدعي جزء كبير (إن لم يكن كامل) البيانات، بالتالي الزمن اللازم لقراءة كل البيانات أكثر أهمية من التأخير الحاصل عند قراءة سجل واحد.

3– العتاد الصلب المعقول: إن Hadoop لا يتطلب عتاد صلب باهظ الثمن. حيث تم تصميمه ليعمل على عناقيد تشتمل على عتاد صلب متوفر بأسعار معقولة ويمكن أن يكون من أكثر من مصنع. إن هذه الخاصية تزيد من احتمال الفشل عبر العنقود. تم تصميم Hadoop بحيث لا يلاحظ المستخدم أي انقطاع في الخدمة في حال وقوع الفشل.

من غير المجدي اختبار التطبيقات التي لا تناسب HDFS في الوقت الراهن وأهم هذه التطبيقات هي:

1– التطبيقات التي تطلب تأخر صغير في الوصول إلى البيانات، من رتبة العشرات من الميلي ثانية، إن هذه التطبيقات لا تعمل بشكل جيد مع HDFS.

2– التطبيقات التي تستخدم العديد من الملفات ذات الحجوم الصغيرة: لأن HDFS يستخدم عقدة مركزية تسمى namenode تقوم بتحميل جميع المعلومات التي تخص الملفات المخزنة في HDFS ضمن الذاكرة.

3– التطبيقات التي تحتاج للكتابة عدة مرات والتعديل العشوائي: إن الملفات في HDFS تتم كتابتها من قبل مصدر واحد، تتم الكتابة بنمط الإضافة على نهاية الملف، لا يوجد (بالصيغة الحالية ل HDFS) دعم للتعديل بمواضع عشوائية ضمن الملف.

و لا نعلم بحدود معرفتنا إن كان سيتم دعم هذه التطبيقات أم لا.

2–3–3 مفاهيم HDFS

2–3–3–1 الكتل Blocks

إن القرص الصلب يحتوي على حجم كتلة يعبر عن أقل كمية بيانات يمكن أن تتم كتابتها أو قراءتها من القرص. إن نظام الملفات الخاص بقرص واحد يعتمد على ذلك حيث يتعامل مع البيانات على أنها تمثل عدد صحيح من الكتل. بالنسبة لنظام الملفات المحلي يكون حجم الكتلة عبارة عن عدد من الكيلو بايتات من الحجم، وبشكل عام يكون حجم الكتلة 512 بايت. إن ذلك يكون شفاف بالنسبة للمستخدم الذي يقوم بالقراءة والكتابة دون أن يفكر بحجم البيانات.

إن HDFS يمتلك نفس مفهوم الكتل، لكن الكتلة في HDFS تشتمل على حجم أكبر هو بشكل افتراضي 128 ميغابايت. وبشكل مشابه لنظام الملفات الخاص بقرص واحد فإن الملفات في HDFS يتم تقسيمها إلى قطع بحيث يكون حجمها مساويا لحجم الكتلة في HDFS، ويتم تخزين كل قطعة بشكل منفصل. بشكل مختلف عن نظام الملفات الخاص بقرص واحد فإن الملفات ذات الحجوم الأصغر من حجم الكتلة لا تستهلك كامل حجم الكتلة (على سبيل المثال، ليكن لدينا ملف حجمه 1 ميغا بايت، يتم تخزينه باستخدام بلوك بحجم 128 ميغابايت، لكنه لا يستهلك إلا 1 ميغابايت من القرص)[2, p. 45].

سبب كبر حجم الكتلة في HDFS

إن حجم الكتل في HDFS يعتبر أكبر من الكتل الخاصة بالقرص الواحد، والهدف هو تقليل زمن الوصول. فكلما زاد حجم الكتلة سيزداد زمن نقل البيانات ليصبح أكبر من زمن الطلب بشكل واضح (seek time). بالتالي نقل ملفات كبيرة تتألف من كتل ذات حجوم كبيرة يمكن أن يكون مساويا تقريبا لسرعة النقل للقرص الصلب.

و بحساب بسيط يمكننا أن نبين انه وعندما يكون زمن الطب مساويا ل10ms وسرعة النقل بحدود 100mbps سيتوجب علينا أن نجعل حجم الكتلة بحدود 100mb، بحيث يكون زمن الطلب يشكل 1% من زمن النقل. في الحقيقة القيمة الافتراضية لحجم الكتلة في HDFS هي 128MB/s. ويمكن تعديل هذه القيمة[5].

إن التجريد الحاصل من استخدام الكتلة في نظام الملفات الموزعة يقدم العديد من الفوائد وأهما [2, p. 46]:

1- يمكن أن يكون حجم الملف أكبر من القدرة التخزينية لأي قرص في الشبكة، بالتالي تقسيم الملف إلى عدة كتل يسمح بتخزينه على عدة أقراص.

2- إن جعل وحدة التجريد تتمثل بالكتلة بدلا من الملف يساهم في تبسيط نظام التخزين. إن التبسيط مكروه في معظم الأنظمة إلا أنه ضروري بشكل خاص من أجل النظم الموزعة التي تتميز بنموذج فشل متأرجح. إن تعامل نظام التخزين مع الكتل يسهل عملية إدارة التخزين وذلك لأن الكتل تتسم بالحجم الثابت والموحد وتسهل عملية حساب القدرة التخزينية للأقراص. كما يتم استبعاد مخاوف الـ metadata لأن الكتل عبارة عن قطع من البيانات الواجب تخزينها أما البيانات الخاصة بالملف مثل سماحيات الوصول واسم الملف (file metadata) لا يتم تخزينها مع الكتل وإنما هنالك نظام آخر مسؤول عن هذه المهمة.

3- إن الكتل مناسبة أكثر لعملية التكرار التي تؤمن تحمل الفشل والتوافرية (availability). ومن أجل التأمين ضد حالات فشل الأقراص أو الحواسيب أو تخرب الكتل، يتم نسخ كل كتلة إلى عدد قليل من الحواسيب المستقلة (عادة ثلاثة). في حال أصبحت الكتلة غير متوفرة تتم قراءة نسخة عنها من موضع آخر بحيث تكون هذه العملية شفافة بالنسبة للمستخدم. في حال أصبحت بعض الكتل غير متوفرة بسبب فشل العقدة أو بسبب التخريب، يتم نسخ الكتلة من مكانها البديل إلى أماكن جديد بحيث يصبح عامل التكرار صحيح (عامل التكرار هو عدد يمثل عدد النسخ لكل كتلة). يمكن أن تتم زيادة عامل التكرار من أجل بعض الملفات التي يتم طلبها كثيرا وذلك لموازنة الحمل ضمن العنقود.

2-3-3-2 عقدة الأسماء وعقدة البيانات (NameNode and DataNode)

إن عنقود HDFS يحتوي على نوعين من العقد التي تعمل بأسلوب المدير والعمال:

- عقدة الأسماء (NameNode) والتي تمثل المدير في العنقود.
- عقد البيانات (DataNodes) وهذه العقد تمثل العمال.

إن عقدة الأسماء تقوم بإدارة فضاء نظام الملفات. حيث تقوم بإدارة شجرة نظام الملفات والبيانات الوصفية (metadata) لكل الملفات والمجلدات في الشجرة. إن هذه المعلومات يتم تخزينها بشكل ثابت على القرص المحلي على شكل ملفين هما:

- ملف يمثل صورة فضاء الأسماء namenode image.
- ملف يمثل سجل التحرير edit log.

كما أن عقدة الأسماء تعلم بعقد البيانات وأماكن تواجد جميع كتل الملفات المخزنة. في الواقع هذه المعلومات تكون في الذاكرة ولا يتم تخذينها بشكل ثابت لأنه يتم إعادة تشكيل هذه البيانات من عقد البيانات عندما يتم تشغيل النظام.

إن عقد البيانات تمثل العمال، تقوم هذه العقد بتخزين وإرجاع الكتل عندما يطلب منهن ذلك (الطلب يتم من قبل الزبون أو عقدة الأسماء)، كما تقوم عقد البيانات بإرسال تقارير دورية إلى عقدة الأسماء تحتوي على قائمة بالكتل الموجودة ضمن هذه العقدة[5].

لا يمكن تشغيل نظام الملفات من دون عقدة الأسماء. وفي الحقيقة إن فشل الآلة التي تشغل عقدة الأسماء سيؤدي إلى ضياع كل الملفات المخزنة في نظام الملفات. بالتالي لا بد من آلية تجعل عقدة الأسماء مقاومة للفشل. إن Hadoop قدم طريقتين لذلك[2, p. 47]:

- الطريقة الأولى: النسخ الاحتياطي للملفات التي تمثل الحالة الثابتة لنظام الملفات. يمكن إعداد Hadoop بحيث يتم كتابة الحالة الثابتة إلى عدة نظم ملفات. بحيث تتم عملية الكتابة بشكل متزامن وذري (atomic) إن الإعداد المعتاد في هذه الحالة هو الكتابة على نظام الملفات المحلي.
- كما يمكن تشغيل عقدة أسماء ثانوية، وتتمثل مهمة هذه العقدة بدمج صورة فضاء الأسماء بسجل التحرير من أجل تجنب النمو الكبير في حجم سجل التحرير. عادة ما يتم تشغيل عقدة الأسماء الثانوية على آلة مستقلة وذلك لأنها تتطلب قدرة حسابية وحجم ذاكري مماثلين للقدرة الحسابية والحجم الذاكري الخاصين بعقدة الاسماء. تقوم هذه العقدة بالاحتفاظ بنسخة من صورة فضاء الأسماء المدموجة، والتي يمكن استخدامها في حال فشل عقدة الأسماء. على كل حال، إن حالة عقدة الأسماء الثانوية تعتبر متأخرة عن حالة عقدة الأسماء وبالتالي وفي حال الفشل الكامل لعقدة الأسماء سيكون فقد البيانات أمرا حتميا ولا بد من استخدام آلية من أجل التوافرية العالية.

2-3-4 التوافرية العالية (HDFS High Availability)

إن حفظ البيانات الوصفية الخاصة بعقدة الأسماء على عدة نظم ملفات واستخدام عقدة الاسماء الثانوية من أجل إنشاء نقاط تفحص من أجل حفظ البيانات من الضياع له دور جيد لمقاومة ضياع البيانات إلى حد ما، إلا أن ذلك لا يؤمن توافرية عالية لنظام الملفات. لأن عقدة الأسماء لا تزال نقطة وحيدة للفشل. وفي حال فشلها فإن جميع الزبائن –بما ضمنهم أعمال المقابلة والاختزال–لن يتمكنوا من القراءة أو الكتابة أو استعراض الملفات، وذلك لأن عقدة الأسماء تمثل المخزن الوحيد للبيانات الوصفية ولعمليات الحصول على الملفات. وفي هكذا حالة سيصبح كامل نظام Hadoop خارج الخدمة حتى يتم تشغيل عقدة أسماء جديدة.

من أجل الاستعادة من الفشل في هكذا حالات، يقوم مدير النظام بتشغيل عقدة أسماء أساسية جديدة بحيث تحتوي على نسخة من البيانات الوصفية ويتم اعدادها واعداد الزبائن من أجل أن يقوموا باستخدامها. لا يمكن لعقدة الأسماء الجديدة أن تقوم بتخديم الزبائن حتى يتحقق التالي:

- أن تقوم هذه العقدة بتحميل صورة فضاء الأسماء إلى ذاكرتها الرئيسية.
- تهيئة سجل التحرير الخاص بها.
- استلام عددا كافيا من تقارير الكتل من عقد البيانات وذلك لتنتقل إلى الحالة الآمنة.

من أجل العناقيد الحاسوبية الكبيرة التي تحتوي على عدد كبير من الملفات والكتل، يكون الوقت اللازم للتشغيل بعد الفشل بحدود ثلاثين دقيقة أو أكثر.

قام Hadoop 2 بمعالجة هذه الحالة من خلال جعل HDFS يدعم التوافرية العالية (High Availability). حيث يتم تشغيل زوج من عقد الأسماء بحيث تكون إحدى هذه العقد فعالة والعقدة الثانية بوضعية الاستعداد (Stand by). وفي حال فشل عقدة الأسماء الفعالة، يتم وضع العقدة التي تعمل بوضع الاستعداد في الخدمة[2, p. 49].

2-3-5 واجهات HDFS

إن Hadoop محقق بلغة جافا بالتالي معظم الواجهات المتوفرة للتعامل مع نظام الملفات الخاص بـHadoop متوافقة مع JAVA API. حيث يعتبر موجه الأوامر الخاص بـ HDFS تطبيق جافا يستخدم الصف FileSystem من أجل تأمين المهام الأساسية الخاصة بنظام الملفات. هنالك العديد من الواجهات الممكن استخدامها للتعامل مع HDFS وأهمها[5]:

1- HTTP

2- FTP Client

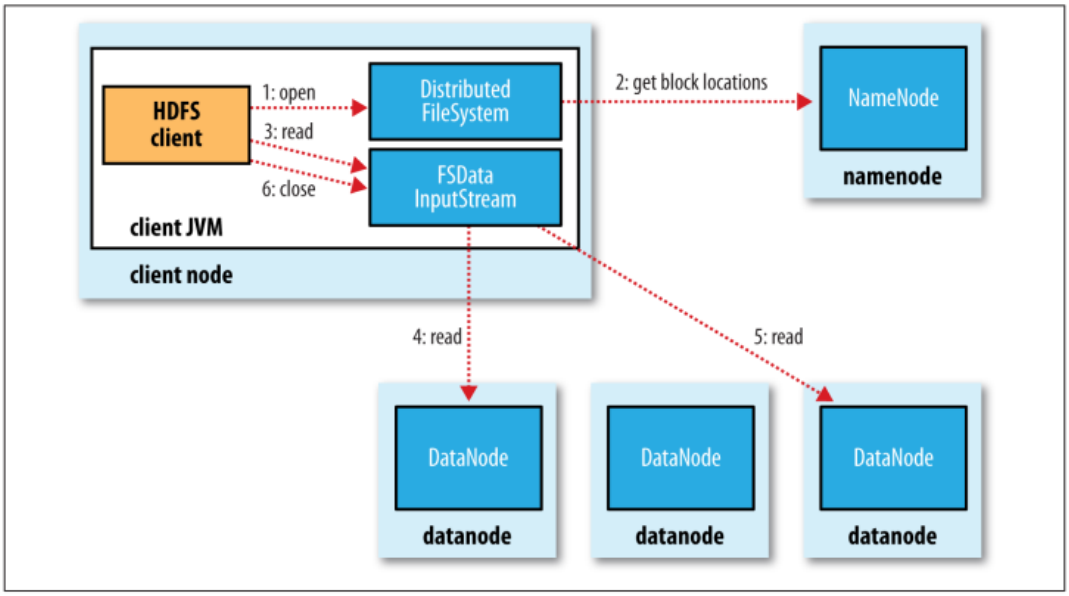
3- File system shell

4- Java interface

2-3-6 تدفق البيانات في HDFS

2-3-6-1 الشكل التفصيلي لقراءة ملف

ليكن لدينا الشكل (2-5) والذي يحتوي على التسلسل الأساسي للأحداث عند قراءة ملف.



الشكل (2-5) قراءة الزبون من HDFS [2, p. 69]

يقوم الزبون بفتح الملف الذي يريد قراءته من خلال استدعاء الدالة open() على الغرض FileSystem، وهذا الغرض هو مثيل من الصف DistributedFileSystem (الخطوة 1 من الشكل(2-5)). يقوم DistributedFileSystem بمخاطبة عقدة الأسماء باستدعاء الطرائق عن بعد (Remote Procedure Call)، وذلك من أجل تحديد أماكن الكتل الأولى في الملف (الخطوة 2). تقوم عقدة الأسماء ومن أجل كل كتلة بإعادة عناوين العقد التي تحتوي على هذه الكتلة. كما أن عقدة الأسماء تقوم بترتيب هذه العناوين بحسب القرب من الزبون (وذلك تبعا لهيكلية الشبكة). في حال كان الزبون يمثل عقدة بيانات (كما قد يحدث في مهام المقابلة والاختزال)، سيقوم الزبون بالقراءة من عقدة البيانات المحلية في حال كانت هذه العقدة تستضيف نسخة من الكتلة.

يقوم الصف DistributedFileSystem بإعادة غرض من النوع FSDataInputStream (وهو عبارة عن مجرى دخل يدعم طلب الملفات) إلى الزبون ليستطيع الأخير القراءة منه. FSDataInputStream يغلف DFSInputStream الذي يقوم بإدارة الدخل والخرج لعقد البيانات وعقدة الأسماء.

يقوم الزبون باستدعاء الدالة ()read على المجرى (الخطوة 3). يقوم DFSInputStream(الذي يخزن عناوين الكتل الأولى من الملف) بالاتصال بأقرب عقدة بيانات تحتوي على أول كتلة من الملف. بعدها يتم نقل البيانات من عقدة البيانات إلى الزبون، يقوم الزبون بشكل متكرر باستدعاء الدالة ()read على المجرى (الخطوة 4). عند الوصول إلى نهاية الكتلة، سيقوم الصف DFSInputStream بإغلاق الاتصال مع عقدة البينات ومن ثم سيقوم بإيجاد أقرب عقدة من أجل الكتلة التالية (الخطوة 5). إن ذلك يتم بشكل شفاف بالنسبة للزبون، والذي بدوره يقوم بالقراءة من مجرى مستمر.

يتم قراءة الكتل بالترتيب، عندما يقوم الصف DFSInputStream بفتح اتصال جديد مع عقدة بيانات يقوم الزبون بالقراءة من خلال هذا المجرى. كما يقوم الزبون أيضا بالاتصال بعقدة الأسماء من أجل الحصول على أماكن عقد البيانات من أجل الكتل اللاحقة. عندما ينتهي الزبون من القراءة يقوم باستدعاء الدالة ()close على الصف DFSInputStream (الخطوة 6).

خلال عملية القراءة، اذا واجه DFSInputStream خطأ أثناء الاتصال بعقدة البيانات، سيقوم بالمحاولة مع ثاني أقرب عقدة تحتوي على نفس الكتلة. كما أنه يتذكر عقد البيانات التي فشلت بالتالي لن يحاول الاتصال بها للحصول على كتل أخرى. كما يقوم الصف DFSInputStream بالتحقق من الـ checksum الخاص بالبيانات التي يتم نقلها من عقدة البيانات. وعند اكتشاف كتل تالفة، يحاول DFSInputStream قراءة الكتلة من عقدة بيانات أخرى، كما يقوم بإرسال تقرير إلى عقدة الاسماء يحدد به الكتلة المخربة.

إن أهم شيء في هذا التصميم هو أن الزبون يقوم بالاتصال مع عقدة الأسماء بشكل مباشر من أجل الحصول على عناوين عقد البيانات التي تحتوي على الكتل. إن ذلك يسمح لـ HDFS بالتوسع ليتم استخدامه من قبل عدد كبير من المستخدمين وذلك لأن البيانات تنتشر ضمن عقد البيانات عبر

العنقود. ولما كانت عقدة الأسماء تقوم بتخديم الزبائن بأماكن الكتل فقط (وهذه المعلومات موجودة في الذاكرة) بالتالي سيكون أداء عقدة الأسماء فعال بشكل كبير.

2-3-6-2 هيكلية الشبكة وHadoop

كيف يمكن أن نحدد قرب وبعد عقدتين يقعان في نفس الشبكة المحلية؟

في سياق معالجة البيانات ذات الحجوم الضخمة سيكون العامل المحدد للقرب هو معدل نقل البينات بين العقد. بالتالي يتم استخدام عرض الحزمة ليكون مقياس المسافة بين العقد.

وعوضا عن قياس عرض الحزمة بين العقد (والذي يمكن أن يكون أمرا صعبا) قام Hadoop باستخدام طريقة بسيطة، حيث اعتبر الشبكة بشكل شجرة والمسافة بين عقدتين هي مجموع أقصر مسافتين بين هاتين العقدتين والسلف المشترك لهما. إن المستويات في الشجرة غير معرفة بشكل مسبق، لكن عادة ما يكون لدينا مستويات مثل (العقدة التي تعمل عليها العملية والرف ومركز البيانات (DataCenter)). الفكرة هي أن عرض الحزمة يتناقص بشكل تدريجي وبالترتيب التالي:

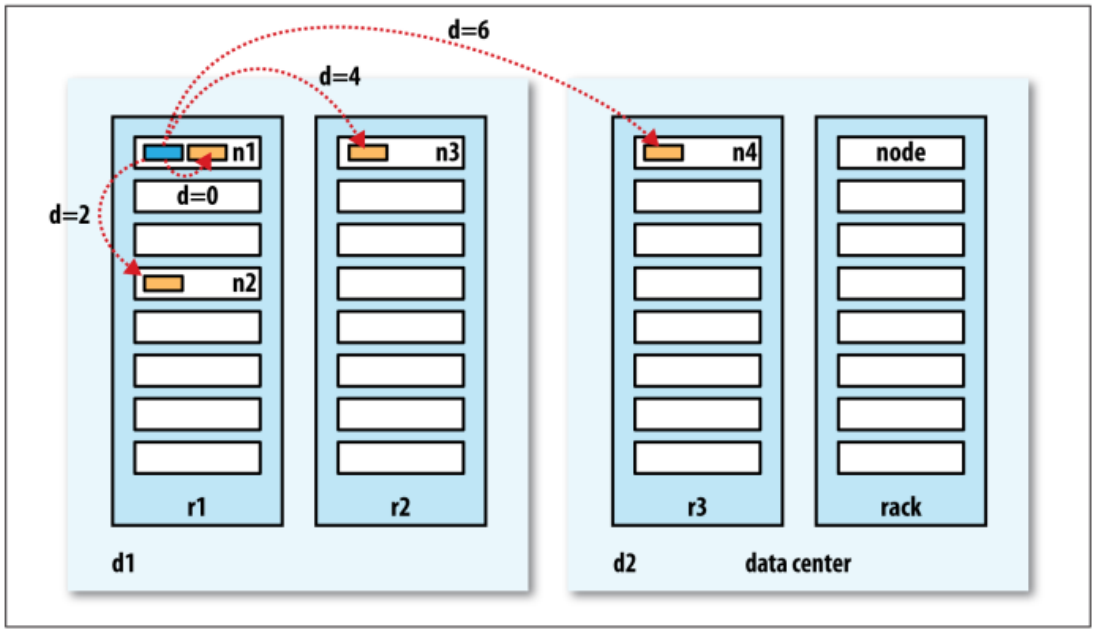
- العملية التي تعمل على العقدة
- العقد المختلفة ضمن نفس الرف
- العقد المختلفة على رفين مختلفين ولكن ضمن نفس مركز البيانات.
- عقد ضمن مراكز بيانات مختلفة.

على سبيل المثال، ليكن لدينا العقدة n1 على الرف r1 ضمن مركز البيانات d1. يمكن تمثيل ذلك بالشكل التالي

/d1/r1/n1. وستكون المسافات بالشكل التالي

- distance(/d1/r1/n1, /d1/r1/n1) = 0 (processes on the same node)
- distance(/d1/r1/n1, /d1/r1/n2) = 2 (different nodes on the same rack)
- distance(/d1/r1/n1, /d1/r2/n3) = 4 (nodes on different racks in the same data center)
- distance(/d1/r1/n1, /d2/r3/n4) = 6 (nodes in different data centers)

هذه العلمية موضحة بالشكل (2–6)



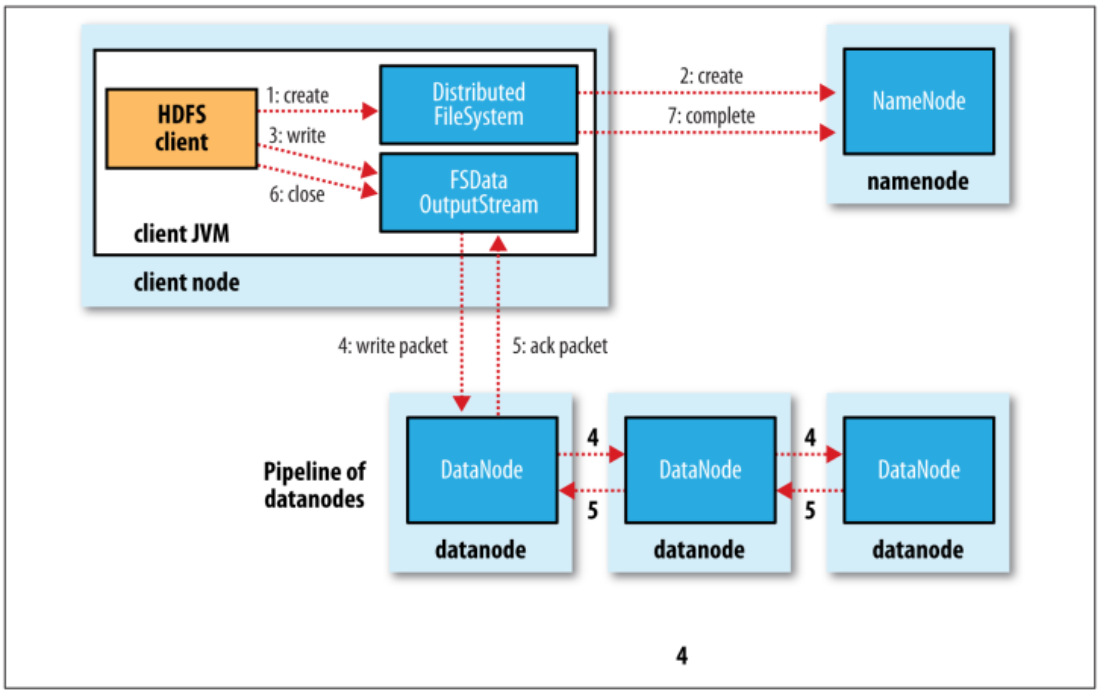
الشكل (2–6) المسافات الشبكية في Hadoop [2, p. 71]

أخيرا، من الضروري ملاحظة أن Hadoop لا يقوم باستكشاف الشبكة بشكل سحري نيابة عنك، فهو بحاجة لبعض المساعدة. وبشكل افتراضي يعتبر أن الشبكة مسطحة اي أن كل العقد تقع في نفس الرف.

2–2–6–3 الشكل التفصيلي لكتابة ملف

سنتكلم الآن عن كيفية كتابة الملفات في HDFS بشيء من التفصيل وهذا يساعد على فهم تدفق البيانات من أجل توضيح النموذج المتماسك لـ HDFS.

سنفترض أننا نريد إنشاء ملف جديد ومن ثم سنقوم بكتابة البيانات عليه بعدها سنقوم بإغلاق الملف. وهذا موضح في الشكل (2–7)



الشكل (2–7) كتابة البيانات على HDFS [2, p. 72]

يقوم الزبون بإنشاء ملف من خلال استدعاء الدالة () create على الصف DistributedFileSystem (الخطوة 1). يقوم الصف DistributedFileSystem باستدعاء من نوع RPC (Remote Procedure Call) على عقدة الأسماء من أجل إنشاء ملف جديد في فضاء الاسماء الخاص بنظام الملفات، وفي البداية لا يرتبط هذا الملف مع أي كتلة (الخطوة 2). تقوم عقدة الأسماء بالعديد من الاختبارات للتأكد من أن الملف غير موجود مسبقا وأن الزبون يمتلك الصلاحيات اللازمة لإنشاء ملف. في حال تم تجاوز هذه الاختبارات بشكل صحيح، تقوم عقدة الأسماء بإضافة سجل يعبر عن الملف الجيد، في حال العكس تفشل علمية الإنشاء ويحصل الزبون على استثناء من النوع IOException.

يرجع الصف DistributedFileSystem بغرض من النوع FSDataOutputStream بحيث يستطيع الزبون البدء بعملية كتابة البيانات. إن الصف FSDataOutputStream يغلف الصف DFSOutputStream الذي يقوم بمعالجة الاتصال مع عقد البيانات وعقدة الأسماء.

حالما يبدأ الزبون بكتابة البيانات (الخطوة 3)، يقوم DFSOutputStream بتقسيم البيانات إلى حزم، والتي يتم كتابتها إلى رتل داخلي يدعى رتل البيانات. يتم استهلاك رتل البيانات من قبل DataStreamer، وهذا الصف يقوم بالطلب من عقدة الأسماء أن تضع كتل جديدة من خلال

اختيار قائمة تحتوي على عقد البيانات المناسبة لتخزين نسخ البيانات. إن قائمة عقد البيانات تشكل خط انبوبي، وهنا سنعتبر مستوى التكرار مساويا لثلاثة، بالتالي هنالك ثلاث عقد.

يقوم الصف DataStreamer بتمرير الحزم إلى أول عقدة بيانات في الخط الأنبوبي(pipeline)، والتي تقوم بتخزين كل حزمة وتمريرها إلى العقدة الثانية في الخط الأنبوبي. وبشكل مشابه، تقوم عقدة البيانات الثانية بتخزين الحزمة وتمريرها إلى العقدة الثالثة والأخيرة في الخط الأنبوبي (الخطوة 4).

يقوم الصف DFSOutputStream بالإبقاء على رتل داخلي من الحزم التي تنتظر إشعار وصولها من عقد البيانات، ويدعى هذا الرتل برتل الإشعارات ack queue. يتم حذف الحزمة من الرتل في حال وصول إشعار من كل عقدة ضمن الخط الأنبوبي (الخطوة 5).

في حال فشل أي عقدة أثناء عملية الكتابة، يتم اتخاذ الإجراءات التالية، وهذه الإجراءات شفافة بالنسبة للمستخدم الذي يقوم بعملية الكتابة. أولا، يتم إغلاق الخط الأنبوبي، وأي حزمة في رتل الإشعار تضاف إلى بداية رتل البيانات بالتالي عقد البيانات التي تلي العقدة الفاشلة في الخط الأنبوبي لن تفقد أي حزمة.

يتم إعطاء الكتلة الحالية في عقد البيانات السليمة معرفا جديدا، بالتالي الكتلة الجزئية الموجودة على العقدة الفاشلة سوف يتم حذفه عندما تتعافى العقدة من الفشل في وقت لاحق.

يتم إزالة العقدة الفاشلة من الخط الأنبوبي، ويتم إنشاء خط أنبوبي جديد يحتوي على العقدتين السليمتين. تتم كتابة ما تبقى من بيانات الكتلة على عقد البيانات السليمة الموجودة في الخط الأنبوبي. تلاحظ عقدة الأسماء أن الكتلة ما تزال تخضع لعملية التكرار، ويتم وضعها من أجل التكرار على عقدة أخرى لاحقا. أما باقي الكتل فيتم التعامل معها بشكل طبيعي.

يمكن أن تتعرض أكثر من عقدة بيانات للفشل عند كتابة الكتلة ولكن احتمال حدوث ذلك هو احتمال ضئيل جدا. ما دامت عملية الكتابة تمت على عدد من النسخ يساوي الخاصية dfs.namenode.replication.min (قيمتها الافتراضية مساوية 1) بالتالي يمكن اعتبار أن عملية الكتابة قد نجحت، وبعدها سيتم تكرار الكتلة بشكل غير متزامن عبر العنقود حتى الوصول إلى عامل التكرار dfs.replication الذي يحوي القيمة 3 بشكل افتراضي.

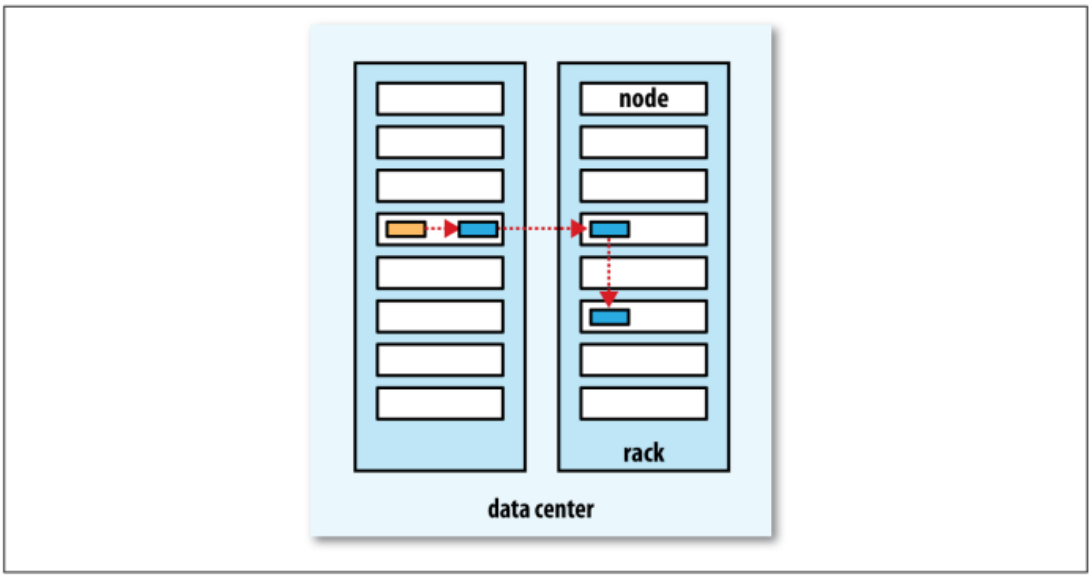
عندما ينتهي الزبون من عملية كتابة البيانات، يقوم باستدعاء الدالة () close على المجرى (الخطوة 6). إن هذا الاستدعاء يقوم بتفريغ كل الحزم وتوجيهها إلى الخط الأنبوبي وينتظر حتى وصول الإشعارات وبعدها يقوم بالاتصال بعقدة الأسماء من أجل إخبارها بأنه تم الانتهاء من كتابة الملف (الخطوة 7). إن عقدة الأسماء تعرف ما هي الكتل التي تنتمي للملف (لأن الصف DataStreamer يطلب أماكن الكتل)، بالتالي عليه أن ينتظر حتى يتم تكرار الكتل قبل أن يعود بالنتيجة الناجحة.

2-3-6-4 أماكن النسخ

كيف تحدد عقدة الأسماء عقد البيانات التي ستقوم بتخزين النسخ؟ هنالك مفاضلة بين الموثوقية وعرض حزمة الكتابة وعرض حزمة القراءة. على سبيل المثال، إن وضع كل النسخ على عقدة واحدة سيقلل من زمن الكتابة (لأن الخط الأنبوبي الخاص بالكتابة يعمل على نفس العقدة)، لكن ذلك لا يؤمن زيادة واقعية في الموثوقية (لأنه وفي حال فشل هذه العقدة بالتالي سيكون هنالك فقد في البيانات). كما أن عرض الحزمة الحقيقي أعلى بين العقد ضمن نفس الرف مقارنة بالعقد التي تنتمي إلى رفوف مختلفة، كما يمكن الوصول إلى موثوقية أكبر في حال تخزين نسخة في مركز بيانات آخر لكن هنالك كلفة إضافية من حيث عرض الحزمة.

إن السياسة الافتراضية لHadoop تقوم على وضع أول نسخة على نفس العقدة التي يعمل عليها الزبون (في حال كان الزبون خارج العنقود يتم اختيار أول عقدة بشكل عشوائي). يتم وضع ثاني نسخة على رف مختلف (off-rack) (الاختيار عشوائي أيضا). أما ثالث نسخة يتم وضعها في نفس الرف الخاص بالنسخة الأولى و لكن على عقدة مختلفة ويتم الاختيار بشكل عشوائي[5] .

حالما يتم تحديد أماكن النسخ، يتم بناء الخط الأنبوبي، مع الأخذ بعين الاعتبار هيكلية الشبكة. ومن أجل معامل تكرار مساويا 3، يمكن أن يكون شكل الخط الأنبوبي مشابه للشكل (2-8)



الشكل(2–8) الشكل النموذجي للخط الأنبوبي للتكرار [2, p. 74]

بشكل عام، إن هذه الاستراتيجية تعطينا موازنة جيدة بين الموثوقية (حيث تمت عملية تخزين البيانات على رفين) وعرض حزمة الكتابة (علمية الكتابة ستعبر من خلال مبدل شبكي واحد) وأداء القراءة (هنالك خيارين بحيث يمكن القراءة من رفين) وعملية نشر الكتل عبر العنقود (حيث قام الزبون بالكتابة على كتلة وحيدة ضمن نفس الرف).

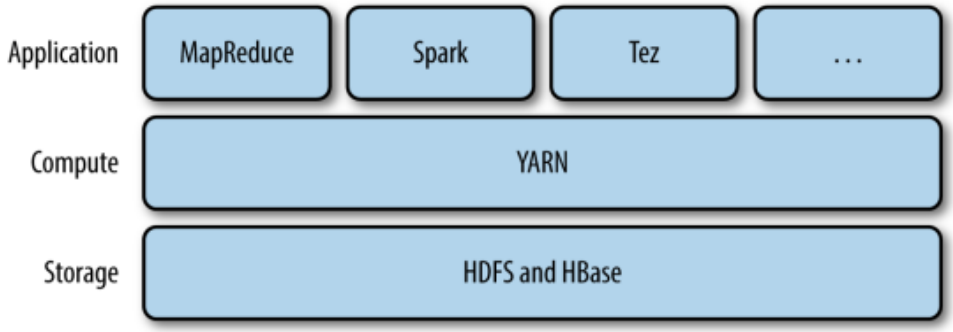
2–3–6–5 الإبقاء على عنقود HDFS متوازنا

يجب أخذ التوازن ضمن العنقود بالحسبان عند نسخ البيانات إلى HDFS. إن HDFS يعمل بشكل أفضل عندما تكون كتل الملف منتشرة بشكل جيد عبر العنقود. إن النسخ الثانية والثالثة منتشرة عبر العنقود لكن النسخة الأولى يمكن أن تسبب عدم توازن وذلك عندما يكون لدينا أكثر من عملية مقابلة على نفس العقدة (في حال كان عدد عمليات المقابلة أكبر من عدد العقد ضمن العنقود) فإذا كان لدينا عقدة تحتوي على عشرين عملية مقابلة بالتالي ستحتوي على 20 كتلة [2, p. 77].

2–4 YARN

إن Apache YARN (Yet Another Resource Negotiator) هو عبارة عن نظام إدارة الموارد لعنقود Hadoop[6]. تم اقتراح YARN من أجل تحسين المقابلة والاختزال في Hadoop 2، لكنه عام بشكل كاف ليدعم أمثلة حسابية موزعة أخرى بشكل جيد.

يقوم YARN بتأمين واجهة برمجية من اجل الطلب والعمل على مصادر العنقود، ولكن من غير المعتاد استخدام هذه الواجهة البرمجية بشكل مباشر. وبدلا عن ذلك، يقوم المستخدم بالكتابة إلى واجهة برمجية ذات مستوا أعلى يتم تأمينها من إطار حساب موزع يعتمد في بنائه على YARN، وبالتالي يتم إخفاء تفاصيل إدارة الموارد عن المستخدم[6]. والشكل (2–9) يوضح هذه الحالة، حيث يبين عدد من الأطر الحسابية الموزعة (Map–Reduce ,spark,tez) والتي تعمل كتطبيق YARN فوق طبقة الحساب (YARN) وفوق طبقة التخزين (HDFS and Hbase).



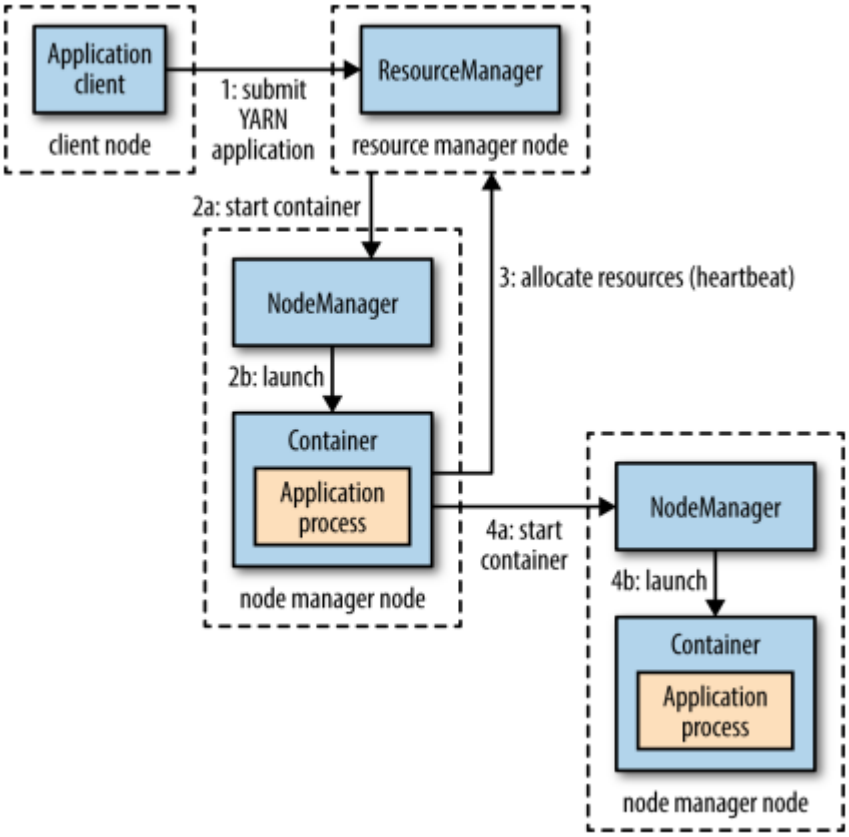
الشكل(2–9) تطبيقات YARN [2, p. 79]

2–4–1 الشكل التفصيلي لعمل تطبيق YARN

يقوم YARNبأداء خدماته الأساسية من خلال تشغيل نوعين من البرامج التي تعمل في الخلفية وهما[6]:

- Resource manager: مدير الموارد وهو واحد بالنسبة للعنقود ويقوم بإدارة استخدام الموارد عبر العنقود الحاسوبي.
- Node manager: يعمل على كل عقدة في العنقود الحاسوبي من أجل تشغيل ومراقبة الحاويات (containers).

تقوم الحاوية بتنفيذ عملية محددة تحتوي على مجموعة من القيود الخاصة بالموارد (الذاكرة والمعالج وغيرها). تبعا لإعدادات YARN يمكن أن تكون الحاوية عبارة عن عملية Unix أو Linux cgroup. الشكل (2–10) يبين كيف يقوم YARN بتشغيل التطبيقات.



الشكل (2–10) كيف يقوم YARN بتشغيل التطبيقات [2, p. 80]

من أجل تشغيل التطبيق على YARN يقوم الزبون بالاتصال بمدير الموارد ويطلب منه تشغيل العملية الرئيسية للتطبيق (application master process) (الخطوة 1). يقوم مدير الموارد بعد ذلك بإيجاد مدير عقدة بإمكانه تشغيل التطبيق الرئيسي ضمن حاوية (الخطوة 2a والخطوة 1b). إن ما يقوم به التطبيق الرئيسي يختلف باختلاف التطبيق. فمن الممكن أن يقوم بتشغيل حساب بسيط ضمن الحاوية التي يعمل ضمنها ويعود بعدها بالنتيجة إلى الزبون. أو يمكنه أن يطلب حاويات أخرى من مدير الموارد (الخطوة 3)، ويستخدم هذه الحاويات من أجل تشغيل حساب موزع (الخطوة 4a والخطوة 4b). وهذا بالضبط ما يقوم به تطبيق YARN من أجل تطبيقات المقابلة والاختزال. يمكن أن نلاحظ ومن خلال الشكل (2–10) أن YARN لا يؤمن التواصل لأي جزء من التطبيق مع باقي الأجزاء (العملية، التطبيق الرئيسي، الزبون). إن معظم تطبيقات YARN الغير بسيطة تستخدم شكل من أشكال التواصل البعيد (مثل طبقة استدعاء الطرائق عن بعد لHadoop ، Hadoop's RPC layer) من أجل تمرير تحديثات الحالة والنتائج للزبون، ولكن ذلك متروك للتطبيق.

2-4-3-1 طلب الموارد

إن YARN يمتلك نموذج مرن من أجل حجز الموارد، إن أي طلب للحاويات يعبر عن مقدار الموارد الحسابية اللازمة لكل حاوية (الذاكرة والمعالج)، ونفس الشيء بالنسبة لقيد تموضع البيانات محليا من أجل الحاويات في الطلب. إن تموضع البيانات محليا (Locality) يضمن أن خوارزميات معالجة البيانات الموزعة تقوم باستخدام عرض الحزمة في العنقود بشكل فعال، لذلك يسمح YARN للتطبيق بتحديد شروط التموضع المحلي بالنسبة للحاويات التي تم طلبها. إن شرط التموضع المحلي يستخدم لطلب حاوية على عقدة محددة أو رف أو أي مكان ضمن العنقود (off–rack).

في بعض الأحيان لا يمكننا تحقيق شرط التموضع المحلي. على سبيل المثال، في حال تم الطلب من عقدة معينة إنشاء حاوية وهي غير قادرة على ذلك (بسبب تشغيل حاويات أخرى)، سيقوم YARN بمحاولة تشغيل حاوية على عقدة أخرى ضمن نفس الرف، أو في أي عقدة ضمن العنقود في حال كان ذلك غير ممكنا.

يمكن لتطبيق YARN أن يقوم بطلب الموارد أثناء عملية التنفيذ، أي يمكن للتطبيق أن يطلب كل الموارد التي يحتاجها من البداية، أو يمكنه أن يتبع طريقة ديناميكية بحيث يطلب المزيد من الموارد بحسب الحاجة.

يقوم Spark باتباع الطريقة الأولى، حيث يبدأ بعدد ثابت من الحاويات. أما إطار المقابلة ولاختزال يمتلك مرحلتين:

- مرحلة المقابلة، حيث يتم طلب الحاويات من البداية.
- مرحلة الاختزال لا يتم طلب الحاويات من البداية.

وفي حال فشل أي مهمة يتم طلب حاويات جديدة بحيث يصبح من الممكن إعادة تشغيل المهام الفاشلة[6].

2-4-2 دورة حياة التطبيق

إن حياة تطبيق YARN يمكن أن تختلف بشكل ديناميكي كبير: ابتداء من التطبيقات قصيرة المدة (بضع ثوان) إلى التطبيقات طويلة المدة (التي يمكن أن تعمل لأيام ويمكن أشهر). بدلا من الانتباه

إلى مقدار طول فترة التطبيق، إنه من الأفضل تصنيف البرامج بحسب الأعمال التي يقوم المستخدمين بتنفيذها. وهنالك ثلاثة نماذج [2, p. 82] :

- النموذج الأول: أن يكون لدينا تطبيق واحد من أجل كل عمل خاص بالمستخدم، وهذا مشابه لتطبيقات المقابلة والاختزال.
- النموذج الثاني، أن يكون لدينا تطبيق واحد من اجل عدة أعمال. وهذه الطريقة فعالة بشكل أكبر من الطريقة الأولى، لأنه من الممكن أن يتم إعادة استخدام الحاويات من قبل الأعمال، وهنالك إمكانية لتخزين البيانات الوسيطة. إن Spark [7] يتبع هذا النوع.
- النموذج الثالث: أن يكون لدينا تطبيق يعمل لفترات طويلة يتم مشاركته من قبل المستخدمين. مثل Impala [8].

2-4-3 مقارنة YARN بـ Map-Reduce 1

إن التحقيق الموزع للمقابلة والاختزال في النسخ الأولى من Hadoop (النسخة 1 وما قبل) يشار إليه على أنه Map-Reduce 1 وذلك من أجل التمييز بينه وبين النسخة الثانية Map-Reduce 2، إن التحقيق الذي يستخدم YARN هو Hadoop 2 والنسخ التي تليه.

في Map-Reduce 1 هنالك برنامجين يعملان في الخلفية يقومان بالتحكم بعملية تنفيذ العمل وهما:

1- Job tracker يقوم بتنسيق الاعمال التي يتم تنفيذها على النظام من خلال جدولة المهام على ال Task trackers.
2- Task tracker يقوم بتنفيذ المهام وإرسال حالة التقدم إلى الـ Job Tracker، حيث يقوم الأخير بالإبقاء على سجلات تبين تقدم كل عمل بشكل عام. وفي حال فشل أي مهمة يقوم Job Tracker بإعادة جدولتها على Task tracker جديد.

في Map-Reduce 1، يقوم الـ Jobtracker بجدولة الأعمال (أي ربط المهام مع الـ Tasktrackers) ومراقبة تقدم المهام (بالإبقاء على أثر للمهام التي تعرضت للفشل أو إعادة الإقلاع أو التنفيذ البطيء كما يقوم بالإحصائيات الخاصة بالمهام مثل العدادات الإجمالية). في YARN يتم تنفيذ هذه المسؤوليات من قبل كيانات منفصلة:

- مدير الموارد
- التطبيق الرئيسي (وهو تطبيق واحد من أجل كل عمل مقابلة واختزال)

إن الـ Jobtracker مسؤول عن تخزين سجل الأعمال التي انتهى تنفيذها، كما هنالك إمكانية لتشغيل مخدم منفصل يعمل كسجل للأعمال. هنالك timeline server في YARN وهو مسؤول عن تخزين سجل التطبيقات.

إن المكافئ لـ Tasktracker في YARN هو مدير العقدة. وفي الجدول(2–1) تلخيص لما تم ذكره في هذه الفقرة.

MapReduce 1	YARN
Jobtracker	Resource manager, application master, timeline server
Tasktracker	Node manager
Slot	Container

الجدول(2–1) مقارنة بين مكونات YARNو Map–Reduce 1 [2, p. 84]

تم تصميم YARNمن أجل حل العديد من المحدودات في Map–Reduce 1. إن أهم الفوائد لاستخدام YARN يمكن تلخيصها بالتالي [2, p. 85]:

- **التوسعية(scalability)**

يمكن أن يتم تشغيل YARNعلى عناقيد ذات حجوم أكبر مقارنة بـ Map–Reduce 1. فمن الممكن أن يتوسع Map–Reduce 1 إلى 4000 عقدة و40000 مهمة، وذلك لأن Jobtracker يقوم بإدارة الأعمال والمهام في نفس الوقت. تمكن YARNمن كسر هذا الحاجز ليصل إلى 10000 عقدة و100000 مهمة من خلال بنيته التي تفصل بين مدير الموارد والتطبيق الرئيسي.

بالمقارنة مع الـ Jobtracker فإن YARN يقوم بتخصيص تطبيق رئيسي بالنسبة لكل عمل والذي يبقى طوال فترة تنفيذ العمل. وهذا النموذج أقرب إلى التصميم الذي تم وضعه للمقابلة والاختزال من قبل google–حيث تقوم العملية الرئيسية بتنسيق مهام المقابلة ومهام الاختزال على العمليات العمال (workers processes) –.

- **التوافرية (Availability)**

يمكن الوصول إلى التوافرية العالية من خلال تكرار الحالة اللازمة لعملية أخرى من أجل إنجاز العمل الضروري لتوفير الخدمة، وذلك عند تعرض العملية المسؤولة عن الخدمة للفشل. على كل حال يعتبر التغير المستمر أمرا يعقد الحالة في ذاكرة Jobtracker (حيث يتم تعديل حالة كل مهمة كل بضع ثوان، على سبيل المثال) وهذا لا يسمح لخدمات الـ Jobtracker بأن تكون متوفرة بشكل عالي.

تم فصل مسؤوليات Jobtracker في YARN بين مدير الموارد والتطبيق الرئيسي، وهذا أدى إلى جعل الخدمة أكثر توافرية .

- **الاستخدامية (Utilization)**

في Map–Reduce 1 يتم إعداد كل Tasktracker بحيث يحتوي على عدد ثابت من الأماكن(slots)، ويتم تقسيم هذه الأماكن إلى أماكن مقابلة وأماكن اختزال خلال وقت الإعداد. لا يمكن استخدام مكان المقابلة إلا من قبل مهام المقابلة، ولا يمكن استخدام أماكن الاختزال إلا من قبل مهام الاختزال.

في YARN، يقوم مدير الموارد بإدارة حوض من الموارد بدلا من استخدام عدد ثابت من الأماكن. إن إطار المقابلة والاختزال الذي يعمل على YARN لا يواجه الحالة التي تضطر فيها مهام الاختزال للانتظار بسبب توفر أماكن مقابلة فقط كما يحصل في Map–Reduce 1. فعند توفر الموارد اللازمة لتشغيل المهمة سيتم منحها للتطبيق.

- **متعدد الإيجار (Multitenancy)**

يمكن أن تكون الفائدة الأهم لـ YARN قدرته على توسيع Hadoop ليشمل تطبيقات موزعة غير المقابلة والاختزال. لأن إطار المقابلة والاختزال هو تطبيق واحد من عدة تطبيقات لـ YARN.

كما أنه من الممكن أن يقوم المستخدمين بتشغيل إصدارات مختلفة من المقابلة والاختزال على نفس عنقود YARN، وهذا ما يجعل عملية ترقية المقابلة والاختزال أمرا أكثر قابلية للإدارة.

بسبب الانتشار الواسع لـ Hadoop 2 بالتالي وعند ذكر مصطلح المقابلة والاختزال (Map–Reduce) خلال الفصول التالية في هذا البحث فإننا نقصد (Map–Reduce 2).

2-4-4 الجدولة في YARN (Scheduling in YARN)

في الحالة المثالية، يتم تخديم الطلبات في YARN بشكل آني. ولكن فعليا وفي حياتنا الواقعية تكون الموارد محدودة، وفي العناقيد المزدحمة بالتالي يتوجب على التطبيقات أن تنتظر حتى يتم تلبية طلبها. هنا تبرز الحاجة لمجدول YARN من أجل إسناد الموارد إلى التطبيقات تبعا لسياسة معرفة بشكل مسبق. إن عملية الجدولة بشكل عام هي عملية صعبة، ولا يوجد اي طريقة للجدول يمكن أن نطلق عليها الطريقة الأفضل على الإطلاق، بالتالي يقوم YARN بتأمين عدة خيرات للجدولة سنتعرف عليها في الفقرات التالية.

هنالك ثلاثة خيرات للجدولة متوفرة في YARN هي:

1- المجدول الذي يعتمد على أسلوب الرتل FIFO (First in First Out)

يقوم المجدول هنا بوضع التطبيقات في رتل ومن ثم تنفيذ هذه التطبيقات بحسب ترتيب الوصول. أولا تتم تلبية طلبات أول تطبيق في الرتل، وحالما يتوفر موارد كافية يتم تلبية الطلب التالي وهكذا.

إن هذا الأسلوب من الجدولة يعتبر سهل الفهم ولا يحتاج أي إعدادات، لكنه غير ملائم للعناقيد التشاركية. لأنه يمكن أن تقوم التطبيقات الكبيرة باستخدام كل الموارد في العنقود وبالتالي يتوجب على باقي التطبيقات الانتظار. بالتالي لا بد من استخدام أساليب جدولة أخرى من أجل العناقيد التشاركية.

2- المجدول السعوي (Capacity scheduler)

يتم تخصيص رتل منفصل من أجل تخديم الأعمال الصغيرة حالما يتم تسليمها، إن هذا الأسلوب يؤثر على الاستخدامية في العنقود وذلك بسبب حجز جزء من الموارد التي قد لا يتم استخدامها، وبالتالي ستتأخر الأعمال الكبيرة مقارنة بالمجدول الذي يعتمد على الرتل[9].

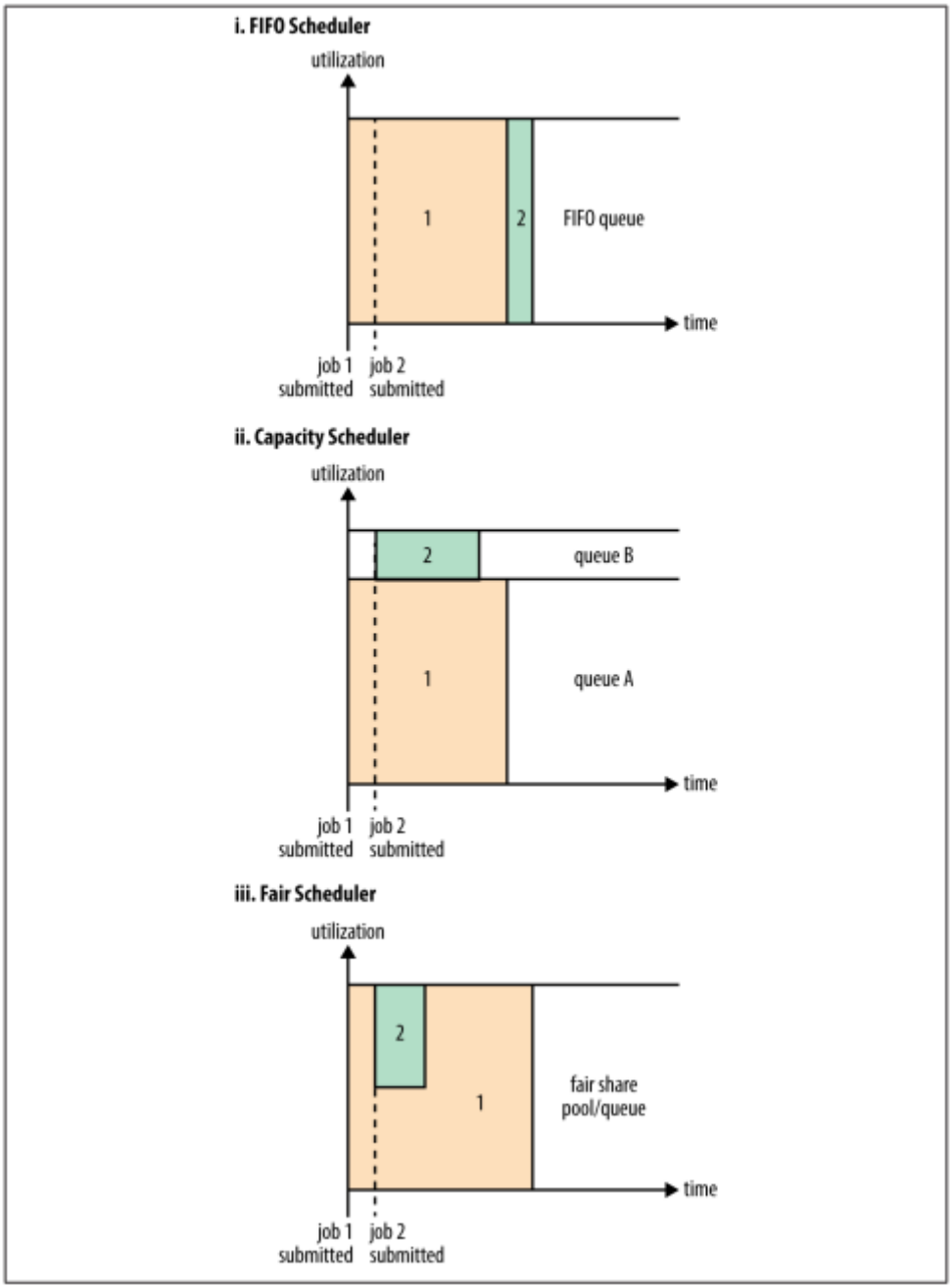
3- المجدول العادل (Fair Scheduler)

ليس هنالك ضرورة لحجز مقدار من سعة العنقود، لأنه وبشكل ديناميكي سيتم موازنة الموارد بين الأعمال التي يتم تنفيذها. فعندما يبدأ تطبيق كبير بالتنفيذ فإنه يستهلك كل الموارد ضمن العنقود وعندما يبدأ تطبيق آخر صغير نسبيا يتم تقاسم الموارد بشكل عادل مناصفتا.

كما يجب الانتباه إلى وجود فترة زمنية تفصل بين اللحظة التي يبدأ فيها التطبيق الثاني واللحظة التي يستحوذ فيها على حصته من الموارد، لأنه يتوجب عليه أن ينتظر انتهاء بعض

الحاويات من تنفيذ المهام المسندة إليها. وبعد انتهاء العمل الثاني (الصغير نسبيا) تصبح الموارد التي يستحوذ عليها متاحة للتطبيق الأول. إن هذه الطريقة تساعد على الاستخدام الأمثل للموارد وتحسن من زمن التنفيذ [10].

إن الشكل (2–11) يبين العمليات الأساسية لأنواع المجدولات الثلاث.



الشكل (2–11) أنواع الجدولة، FIFO I,CAPACITY II,FAIR III [2, p. 87]

إعدادات المجدول السعوي

إن المجدول السعوي يسمح بمشاركة عنقود Hadoop بين عدة منظمات، حيث يتم تخصيص كل منظمة بجزء من سعة العنقود. يتم إعداد كل منظمة برتل مخصص تسند له جزء من موارد العنقود. كما يمكن أن يتم تقسيم الأرتال فيما بعد بحيث نحصل على شكل هرمي، بحيث تتمكن كل منظمة من جعل مجموعات العمل والمستخدمين ضمنها يتشاركون الرتل الخاص بالمنظمة. ضمن كل رتل يتم ترتيب التطبيقات بأسلوب الداخل أولا خارج أولا.

وكما لاحظنا في الشكل (2–11) لا يمكن للعمل الواحد أن يستهلك موارد أكثر من الموارد المخصصة للرتل. وفي حال وجود أكثر من عمل ضمن الرتل و كان هنالك موارد شاغرة، يمكن للمجدول السعوي أن يقوم بإسناد الموارد الشاغرة لأعمال ضمن الرتل، ويدعى هذا السلوك بمرونة الرتل.

2–5 آلية عمل map reduce

في هذا الجزء، سنتكلم عن كيفية عمل المقابلة والاختزال في إطار Hadoop بشيء من التفصيل. إن هذه المعرفة ضرورية لبناء محاك يراعي تفاصيل وضوابط المقابلة والاختزال.

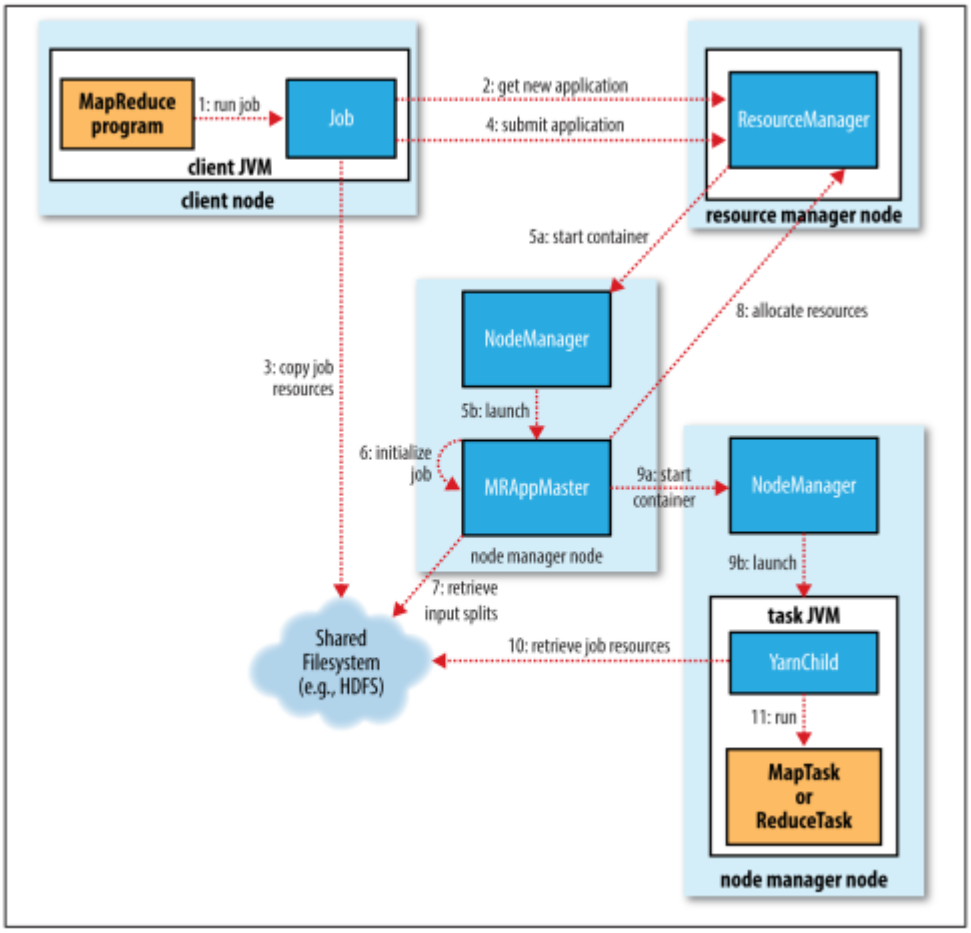
2–5–1 آلية تنفيذ عمل map reduce

يمكننا تشغيل مهمة المقابلة والاختزال من خلال استدعاء دالة ()submit على غرض العمل ،كما يمكننا استدعاء الدالة ()waitForCompletion والتي تقوم بتسليم المهمة اذا كانت غير مسلمة بشكل مسبق وتنتظر هذه الدالة حتى يتم الانتهاء من تنفيذ المهمة [11].

خلف هذا الاستدعاء هنالك كمية كبيرة من المعالجة لا تظهر للعيان. إن الشكل (2–12) يوضح كامل العملية بدرجة عالية من التجريد، إن عملية معالجة المهام باستخدام نموذج المقابلة والاختزال يحتوي على خمسة كيانات منفصلة[2, p. 185]:

- الزبون الذي يقوم بإرسال مهمة المقابلة والاختزال
- YARN مدير الموارد، والذي ينسق تخصيص الموارد الحاسوبية على العنقود.
- YARN مدراء العقد، حيث يقوم كل مدير عقدة بإنشاء ومراقبة حاويات الحساب على الطرفيات في العقود الحاسوبي.

- التطبيق الرئيسي في المقابلة والاختزال والذي يلعب دور السيد، ويقوم بتنسيق المهام التي تنفذ عمل المقابلة والاختزال. يتم تشغيل وجدولة مهمة التطبيق الرئيسي ومهام المقابلة والاختزال من قبل مدير الموارد وتدار هذه المهام من قبل مدراء العقد.
- نظام الملفات الموزع (عادة HDFS) والذي يستخدم لمشاركة ملفات الأعمال بين الكيانات.



الشكل 2–12 كيف يقوم Hadoop بتشغيل عمل المقابلة والاختزال [2, p. 186]

2–5–2 تسليم العمل

إن الدالة submit() تقوم بإنشاء مثيل من النوع JobSubmitter وتقوم باستدعاء الدالة submitJobInternal() عليه (الخطوة 1 في الشكل 2–12). بعد أن يتم تسليم العمل تقوم الدالة waitForCompletion() بشكل متكرر وكل ثانية بطباعة مقدار التقدم في العمل (في حال كان هنالك تقدم) . وعندما يتم الانتهاء من العمل بشكل ناجح، يتم عرض عدادات العمل. أو يتم تسجيل الخطأ الذي سبب فشل العمل.

إن عملية تسليم العمل تم تحقيقها من خلال JobSubmitter والذي يقوم بالتالي:

- يسأل مدير الموارد عن معرف لتطبيق جديد وذلك ليستخدم في عمل مقابلة واختزال جديد (الخطوة 2).
- تفحص التوصيف الخاص بخرج العمل. على سبيل المثال، إذا كان مجلد الخرج غير محدد أو موجود، لن يتم تسليم العمل وسيتم رمي خطأ إلى برنامج المقابلة والاختزال.
- حساب أجزاء الدخل الخاصة بالعمل. إذا لم يكن بالإمكان حساب أجزاء الدخل (على سبيل المثال، مسار الدخل غير موجود) لن يتم تسليم العمل وسيتم رمي خطأ إلى برنامج المقابلة والاختزال.
- نسخ المصادر الضرورية لتشغيل العمل، (والتي تشمل ملف الـ JAR الخاص بالعمل وملف الإعداد وأجزاء الدخل المحسوبة) إلى نظام الملفات التشاركي في مجلد يبدأ اسمه بمعرف العمل (الخطوة 3). يتم نسخ ملف الـ JAR بمعامل تكرار عال (يتم التحكم يهذا المعامل من خلال الخاصية MapReduce.client.sumbit.file.replication) و باعتبار القيمة الافتراضية 10، بالتالي هنالك العديد من النسخ في العنقود يمكن لمدراء العقد أن تصل إليها عندما تقوم بتشغيل مهام من أجل العمل.
- تسليم العمل من خلال استدعاء submitApplication() على مدير الموارد (الخطوة 4)

2-5-3 تهيئة العمل

عندما يتلقى مدير الموارد استدعاء لدالته submitApplication() يقوم بتمرير الاستدعاء إلى مجدول YARN، يقوم المجدول بإسناد حاوية بحيث يقوم مدير الموارد بعدها بتشغيل العملية المسؤولة عن التطبيق الرئيسي تحت إدارة مدير العقدة (الخطوة 5a and 5b).

إن التطبيق الرئيسي لأعمال المقابلة والاختزال هو تطبيق جافا يكون الصف الرئيسي فيه MRAppMaster. يقوم الصف الرئيسي بتهيئة العمل من خلال إنشاء عدد من أغراض المحاسبة من أجل حفظ مسار لتقدم العمل وذلك تبعاً لتقارير التقدم والانتهاء الواردة من المهام (الخطوة 6). بعدها يقوم باستقبال أجزاء الدخل (التي تم حسابها من قبل المستخدم) من نظام الملفات الموزع (الخطوة 7). بعدها يقوم بإنشاء غرض يمثل مهمة المقابلة من أجل كل جزء، كما أن عدد الأغراض التي تمثل عمليات الاختزال يحدد من خلال الخاصية MapReduce .job.reduces (و التي يتم ضبطها من خلال الدالة setNumReduceTasks() الخاصة بالعمل).

يجب على التطبيق الرئيسي أن يقرر كيف سيقوم بتنفيذ المهام التي تشكل عمل المقابلة والاختزال، فإذا كان العمل صغيرا، ربما سيقرر التطبيق الرئيسي أن ينفذ المهام على نفس آلة الجافا الافتراضية JVM. قد يحدث ذلك إذا وجد التطبيق الرئيسي أن كلفة تشغيل المهام في حاويات جديدة –بشكل متفرع بغية الربح في وقت التنفيذ–أكبر من وقت التنفيذ التسلسلي على عقدة واحدة. يطلق على الأعمال التي تندرج تحت هذا النوع uberized أو أن المهمة التي يتم تنفيذها تدعى مهمة uber[1].

يمكن اعتبار الأعمال على أنها صغيرة إذا كان عدد المقابلات (mappers) أقل من عشرة وكان لدينا مختزل واحد (reducer) كما أن حجم الدخل أقل من حجم الكتلة في نظام الملفات الموزع (HDFS).

ملاحظة : يمكن أن يتم تغيير هذه القيم من خلال التعديل على MapReduce .job.ubertask.maxmaps و .job.ubertask.maxreduces MapReduce و reduce.job.ubertask.maxbytes [11].

يجب السماح بالمهام من نوع uber بشكل صريح (إما من أجل أعمال محددة أو عبر العنقود) وذلك من خلال إعطاء القيمة true للخاصية MapReduce .job.ubertask.enable.

أخيرا، وقبل أن يتم تنفيذ أي مهمة، على البرنامج الرئيسي يجب أن يقوم باستدعاء الدالة setupJob() على الصف OutputCommitter.ForFileOutputCommitter ، والذي سيقوم بشكل افتراضي بإنشاء مجلد الخرج النهائي وفضاء العمل المؤقت الخاص بخرج المهمة.

2-5-4 تسليم المهمة

إذا كان العمل غير مناسب ليتم تنفيذه كمهمة uber، فإن التطبيق الرئيسي سيطلب حاويات لكل مهام المقابلة والاختزال الخاصة بالعمل وسيتم الطلب من مدير الموارد(الخطوة 8). يتم الطلب في البداية من أجل مهام المقابلة وبأولوية أعلى من مهام الاختزال، لأن كل مهام المقابلة يجب أن تكمل عملها

[1] uber هي شركة في مجال خدمات النقل قدمت مفهوم جديد في التخديم يعتمد على جعل الخدمة متقطعة يتم طلبها حسب الحاجة

قبل مرحلة الترتيب التي تبدأ بها مهام الاختزال. لا يتم طلب مهام اختزال حتى تنتهي 5% من مهام المقابلة من عملها.

يمكن أن يتم تنفيذ مهام الاختزال في أي مكان في العنقود، لكن المجدول يحاول احترام الطلبات لمهام المقابلة التي تمتلك قيود على البيانات المحلية. في الحالة المثالية تكون المهمة تحتوي على البيانات محليا، أو يمكن أن تكون البيانات الخاصة بالمهمة متموضعة في نفس الرف، أو يمكن أن تكون ليست محلية ولا في نفس الرف وإنما متواجدة في رف آخر غير الرف الحاوي على المهمة التي تقوم بالتنفيذ[11].

كما أن الطلبات تحدد متطلبات المهام من حيث وحدة المعالجة المركزية والذاكرة. بشكل افتراضي يتم اسناد 1024 MB من الذاكرة ونواة افتراضية لكل مهمة مقابلة أو مهمة اختزال. وهذه القيم يمكن التحكم بها بالنسبة لكل عمل وذلك باستخدام الخصائص التالية:

- MapReduce .map.memory.mb
- MapReduce .reduce.memory.mb
- MapReduce .map.cpu.vcores
- MapReduce.reduce.cpu.vcores

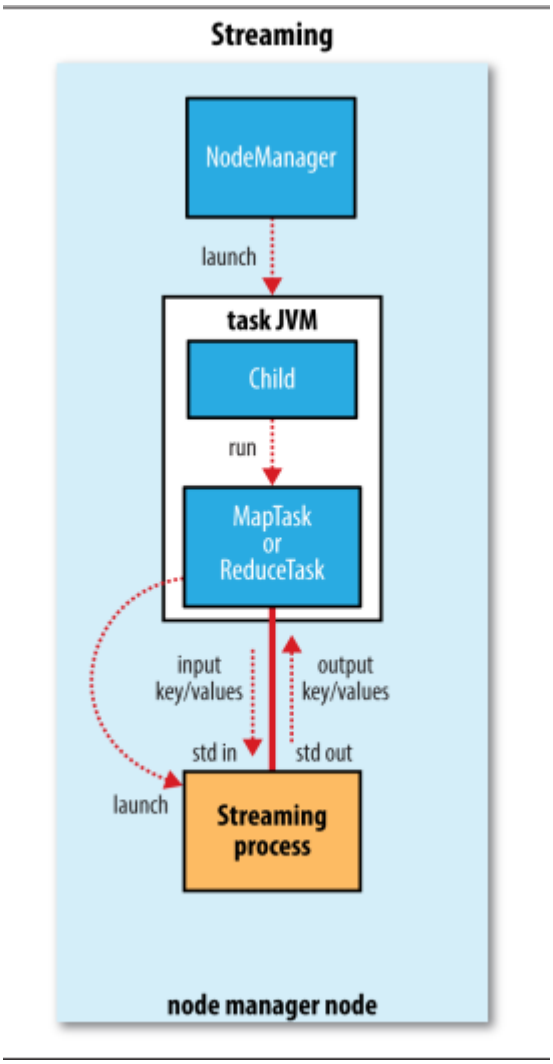
2-5-5 تنفيذ المهمة

حالما يقوم مجدول مدير الموارد بإسناد الموارد المهمة إلى حاوية ضمن عقدة معينة، يقوم التطبيق الرئيسي بتشغيل الحاوية من خلال الاتصال بمدير العقدة (الخطوة 9a and 9b). يتم تنفيذ المهمة من خلال تطبيق جافا يملك صف رئيسي يدعى YarnChild. لكن وقبل أن يبدأ هذا الصف بتنفيذ المهمة فإنه يقوم بتحديد مكان الموارد اللازمة لتنفيذ المهمة (اعدادات العمل وملف الـJAR وأي ملف من الكاش الموزع (الخطوة 10)). بعدها يقوم بتنفيذ مهمة المقابلة أو مهمة الاختزال (الخطوة 11).

يتم تشغيل الصف YarnChild في آلة جافا افتراضية مخصصة له، بالتالي فإن الأخطاء في دوال المستخدم الخاصة بالمقابلة والاختزال أو في الصف YarnChild لن تؤثر على مدير العقدة ولن تسبب له الفشل.

2-5-6 التدفق

إن تقنية التدفق(Streaming) تقوم بتشغيل مهام مقابلة واختزال خاصة يمكنها أن تتواصل وتتلقى بيانات من المستخدم. تقوم مهمة التدفق بالتواصل مع العملية (والتي يمكن أن تكون مكتوبة بأي لغة) باستخدام قنوات دخل وخرج معيارية. خلال تنفيذ المهمة، تقوم عملية الجافا بتمرير أزواج مفتاح–قيمة للعملية الخارجية، والتي تقوم بتنفيذ هذه الأزواج من خلال توابع مقابلة واختزال معرفة من قبل المستخدم وبعدها تقوم بتمرير أزواج الخرج مفتاح–قيمة إلى عملية الجافا. من وجهة نظر مدير العقدة فإن العقدة الأبن تقوم بتنفيذ شيفرة المقابلة والاختزال الخاصة بها.



الشكل (2–13) علاقة التنفيذ التدفقي ومدير العقدة وحاوية المهمة [2, p. 190] .

2-5-7 التقدم وتعديل الحالة

يعرف التنفيذ الدفعي (Batch processing) على أنه تنفيذ سلسلة من الأعمال على الحاسب بشكل غير تفاعلي وبالنسبة لنموذج المقابلة والاختزال فإن التنفيذ الدفعي يعني معالجة حجوم كبيرة من البيانات التي تم تخزينها عبر فترة من الزمن[2, p. 190].

إن أعمال المقابلة والاختزال تقوم بتنفيذ الأعمال الدفعية بحيث يتراوح زمن التنفيذ بين عشرات الثواني إلى عدد من الساعات. ولا بد أن نوضح للمستخدم اسلوب المعالجة لتبيان طول فترة التنفيذ. إن أي عمل (أو أي مهمة خاصة بهذا العمل) يمتلك حالة (status) والتي تحتوي على قيم توضح حالة المهمة أو العمل (مثل: التنفيذ، الفشل، نجاح العمل، تقدم عملية المقابلة والاختزال، قيم عدادات الأعمال، حالة الرسالة أو شرح عنها والذي يمكن تحديده من قبل المستخدم). هذه الحالات تتغير مع سياق العمل، بالتالي ما هي الآلية المتبعة لإيصال هذه القيم للمستخدم؟ عندما تكون المهمة قيد التنفيذ، فإنها تبقي على أثر لتقدمها (نسبة اكتمال المهمة). من أجل مهام المقابلة، نسبة الاكتمال هي نسبة الدخل الذي تم معالجته. لكن تحديد نسبة اكتمال التنفيذ لمهام الاختزال أكثر تعقيدا، لكن النظام يبقى قادرا على توقع نسبة الدخل الذي تم معالجته في مهمة الاختزال. ويتم ذلك من خلال تقسيم التقدم الكلي على ثلاث أجزاء متعلقة بثلاث مراحل (النسخ والترتيب و الاختزال) على سبيل المثال، إذا تم تشغيل الاختزال على نصف دخلها، بالتالي تقدم العملية سيكون 6/5 لأنها أكمله مراحل النسخ والترتيب (3/1 من أجل كل مرحلة) وهي في منتصف مرحلة الاختزال (6/1) أي:

$$\frac{1}{3}+\frac{1}{3}+\frac{1}{6}=\frac{5}{6}$$

من غير الممكن قياس التقدم بشكل دائم، وعلى الرغم من ذلك فإن التقدم يبين لـ(Hadoop) بأن المهمة تقوم بعمل ما. على سبيل المثال، ليكن لدينا مهمة تقوم بكتابة سجلات الخرج، وعلى الرغم من عدم قدرة هذه المهمة على تحديد نسبة التقدم بالنسبة لعدد السجلات الكلي، فإن التقدم مهم لأنه يمنع Hadoop من اعتبار هذه المهمة فاشلة كونها تقوم بالتقدم.

العمليات التالية تبين أن المهمة تتقدم

- قراءة سجل دخل (في مهمة المقابلة أو الاختزال)
- كتابة سجل خرج (في مهمة المقابلة أو الاختزال)

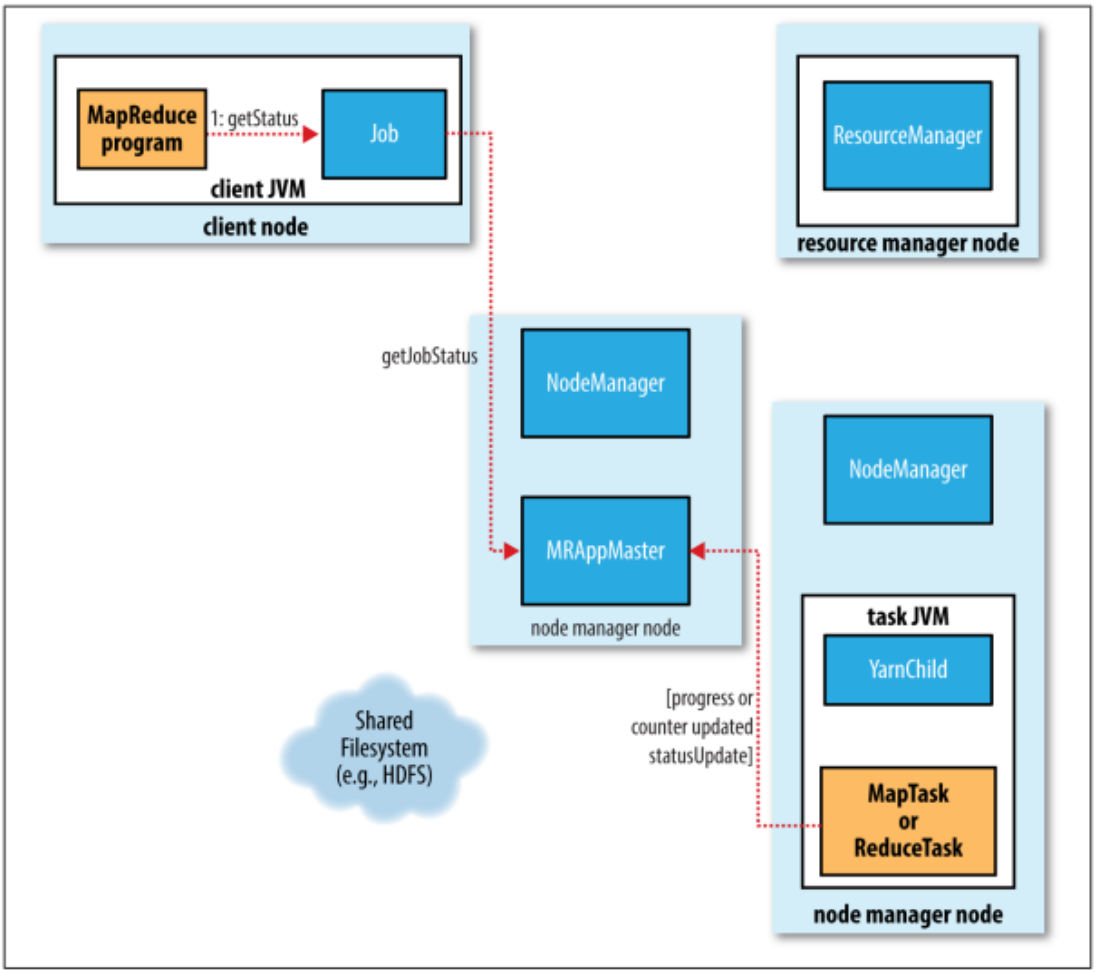
- التعديل على واصف الحالة (من خلال Reporter أو الدالة setStatus() في الصف TaskAttemptContext)
- زيادة عداد(باستخدام الدالة incrCounter في الصف Reporter أو الدالة increment في الصف Counter)
- استدعاء Reporter أو الدالة progress في الصف TaskAttemptContext

كما أن المهمة تحتوي على عدادات تقوم بحساب عدة أحداث مختلفة أثناء عملية تنفيذ المهمة، وهنالك نوعين لهذه العدادات:

- عدادات مبنية داخليا ضمن الإطار مثل عدد سجلات خرج المقابلة.
- عدادات معرفة من قبل المستخدم.

إن واجهة الويب الموجهة للمستخدم والخاصة بمدير الموارد تقوم بإظهار كل التطبيقات(الأعمال) التي يتم تنفيذها، بحيث يظهر بجانب كل تطبيق رابط إلى واجهة الويب الموجهة للمستخدم والخاصة بالتطبيق الرئيسي لهذا التطبيق، بحيث تظهر تفاصيل أكثر متعلقة بعمل المقابلة والاختزال وبتقدم عملية التنفيذ.

في سياق تنفيذ العمل يستلم الزبون آخر تحديثات الحالة من خلال استطلاع هذه القيم من التطبيق الرئيسي كل ثانية (يمكن تحديد هذه الفترة من خلال MapReduce client.progressmonitor.pollinterval.). ويمكن للزبون أن يستخدم الدالة getStatus() من أجل الحصول على نسخة من الصف JobStatus والتي تحتوي على كل معلومات الحالة الخاصة بالعمل والشكل 2–14 يبين كامل العملية.



الشكل 2–14 كيف تتم عملية تحديث وتوليد الحالة عبر نظام المقابلة والاختزال [2, p. 190].

2–5–8 إتمام العمل

عندما يستلم التطبيق الرئيسي تنبيه يبين أنه تم الانتهاء من تنفيذ آخر مهمة من العمل، يقوم عندها التطبيق الرئيسي بتعديل حالة العمل إلىsuccessful ، بعدها وعندما تطلب حالة العمل والتي تبين أن العمل اكتمل بنجاح، سيقوم التطبيق الرئيسي بطباعة رسالة تخبر المستخدم بذلك ويعود من تنفيذ الدالة waitForCompletion(). تتم عملية طباعة العدادات والإحصائيات للخرج عند هذه النقطة.

أخيرا وعند اكتمال العمل، يقوم التطبيق الرئيسي وحاويات المهام بمسح حالة عملهم (بالتالي يتم حذف الخرج الوسيط)، ويتم استدعاء الدالة commitJob() الخاصة بالصف OutputCommitter. تتم أرشفة معلومات العمل من قبل مخدم السجل من أجل السماح للمستخدم بالاستفسار عن معلومات تخص العمل[2, p. 192].

2-5-9 الفشل

في الحقيقة إن أي نص برمجي يقوم المستخدم بكتابته فهو عرضة للأخطاء كما أنه معرض لفشل العمليات وفشل العتاد الصلب. إن القدرة على معالجة مثل هذه الأنواع من الفشل يعتبر من أهم الإيجابيات من استخدام Hadoop بالتالي يمكن للعمل أن يتم تنفيذه وإكماله بشكل صحيح. علينا توقع الفشل لأي من الكينونات التالي[2, pp. 193–196]:

- المهمة
- التطبيق الرئيسي
- مدير العقدة
- مدير الموارد

2-5-9-1 فشل المهمة

إن السبب الشائع لحدوث هذا النوع من الفشل هو الاستثناءات أثناء التنفيذ (runtime exception) التي ترميها شيفرة المستخدم البرمجية الخاصة بمهام المقابلة أو الاختزال. إذا حدث مثل هذا النوع من الفشل، تقوم آلة الجافا الافتراضية للمهمة (task JVM) بإرسال تقرير بالخطأ إلى التطبيق الرئيسي قبل أن تنهي التنفيذ وتغادر. وأخيرا يتم تسجيل الخطأ في سجل المستخدم. يقوم التطبيق الرئيسي باعتبار محاولة المهمة فاشلة (failed) ويقوم بتفريغ الحاوية وتصبح الموارد الخاصة بها متاحة لمهمة أخرى.

من أجل المهام التدفقية، إذا أنهت المهمة التدفقية تنفيذها وخرجت برمز مغاير للصفر، تعتبر كمهمة فاشلة. هذا السلوك يتم التحكم به من قبل الخاصية stream.non.zero.exit.is.failure (القيمة الافتراضية هي true).

هنالك نموذج آخر من الفشل هو الخروج المفاجئ لآلة الجافا الافتراضية الخاصة بالمهمة–لربما كان هنالك خطأ في JVM سبب خروجها من أجل بعض الظروف المفروضة من قبل شيفرة المستخدم البرمجية الخاصة بالمقابلة والاختزال، في هذه الحالة يلاحظ مدير العقدة حالة فشل العقدة ويقوم بإعلام التطبيق الرئيسي بذلك فيقوم الأخير بوسم المحاولة بالفاشلة.

المهام المعلقة يتم التعامل معها بشكل مختلف. حيث يلاحظ التطبيق الرئيسي عدم وصول أي تحديث عن تقدم هذه المهام لفترة من الزمن بعدها يقوم بوسم هذه المهام على أنها فاشلة. إن عملية آلة الجافا

الافتراضية الخاصة بالمهمة يتم إنهائها بشكل أوتوماتيكي بعد هذه الفترة. فترة وقت الخروج (time out) والتي بانقضائها تعتبر المهمة فاشلة هي عشر دقائق بشكل افتراضي ويمكن أن يتم إعدادها من أجل كل عمل أو من أجل العنقود وذلك بإعداد الخاصية MapReduce .task.timeout وإعطائها قيمة بالميلي ثانية.

عند إعطاء القيمة صفر لوقت الخروج يتم تعطيل خاصية وقت الخروج، بالتالي لن يتم اعتبار المهام التي تستهلك وقت طويل أثناء تنفيذها مهام فاشلة. في هذه الحالة ولن تقوم المهام المعلقة بتفريغ الحاويات الخاصة بها، ومع الوقت سينهار العنقود كنتيجة لذلك. بالتالي يجب تجنب هذه الطريقة والتأكد من أن المهمة تقوم بإرسال تقارير التقدم بشكل دوري وكاف.

عندما يتم تنبيه التطبيق الرئيسي بفشل المهمة، يقوم بجدولة التنفيذ الخاص بها من جديد، سيتجنب التطبيق الرئيسي إعادة جدولة المهمة على نفس مدير العقدة الذي فشلت عليه سابقا. في حال فشلت العقدة أربع مرات، لن تتم جدولتها مرة أخرى. هذه القيمة يمكن التحكم بها. إن عدد المحاولات لتنفيذ مهمة يتم التحكم به من خلال الخاصية MapReduce .map.maxattempts بالنسبة لمهام المقابلة ومن خلال الخاصية MapReduce .reduce.maxattempts بالنسبة لمهام الاختزال. بشكل افتراضي وعند فشل أي مهمة أربع مرات (أو أن المهمة فشلت بالعدد المحدد لمرات الفشل) سيفشل العمل بشكل كامل.

من أجل بعض التطبيقات، إنه من غير المناسب ترك العمل عند فشل بعض المهام، حيث من الممكن الاستفادة من النتائج بغض النظر عن الفشل. في هذه الحالة، يتم تحديد نسبة المهام المسموح بفشلها دون أن يؤدي ذلك إلى فشل العمل بشكل كامل، هذه النسبة يتم تحديدها بالنسبة لمهام المقابلة من خلال الخاصية MapReduce .map.failures.maxpercent وبالنسبة لمهام الاختزال من خلال الخاصية MapReduce .reduce.failures.maxpercent.

إن محاولة تنفيذ المهمة يمكن أن يتم إنهائها، وهذا مختلف عن الفشل. يتم إنهاء المهمة في حال كان هنالك نسخة تخمينية أو أن مدير العقدة يعمل بشكل فاشل وبالتالي قام التطبيق الرئيسي باعتبار كل المهام التي يتم تنفيذها عليه يجب انهائها.

إن عدد المهام التي تم إنهائها لا يرتبط بعدد المحاولات المحدد لتنفيذ مهمة.

كما أن المستخدم يمكن أن ينهي أو يفشل محاولة تنفيذ مهمة من خلال واجهة الويب أو من خلال موجه الأوامر باستخدام الأمر (mapred job) من أجل عرض الخيارات الممكنة. كما يمكن إنهاء الأعمال بنفس الطريقة[2, p. 194].

2-5-9-2 فشل التطبيق الرئيسي

كما هو الحال من أجل مهام المقابلة والاختزال من حيث عدد محاولات التنفيذ للوصول إلى التنفيذ الناجح (لمواجهة فشل العتاد الحاسوبي أو الشبكي) فإن التطبيقات في YARN تتم إعادة تنفيذها في حال الفشل. يمكن التحكم بعدد مرات الفشل المسموحة لتنفيذ التطبيق الرئيسي لعمل المقابلة والاختزال من خلال الخاصية MapReduce .am.max-attempts. القيمة الافتراضية هي 2، بالتالي عند فشل تطبيق رئيسي مرتين سيفشل العمل ولن يكون هنالك محاولة ثالثة لتنفيذ التطبيق.

إن YARN يفترض حد لعدد المحاولات الأعظمي من أجل أي تطبيق رئيسي يتم تنفيذه على العنقود، وأي تطبيق لا يمكنه أن يتجاوز هذا الحد. يمكن تعديل هذا الحد من خلال الخاصية yarn.resourcemanager.am.max-attempts حيث القيمة الافتراضية لها هي 2، بالتالي واذا أردنا أن نزيد عدد المحاولات لتطبيقات المقابلة والاختزال الرئيسية، علينا أيضا أن نزيد في إعدادات YARNعلى العنقود.

إن الطريقة المتبعة لاسترجاع العمل تتم كالتالي: يقوم التطبيق الرئيسي بإرسال نبضات إلى مدير الموارد، وفي حال حدوث فشل للتطبيق الرئيسي، سيكتشف مدير الموارد ذلك وسيقوم بإنشاء مثيل جديد من التطبيق الرئيسي ليتم تنفيذه على حاوية جديدة. في هذه الحالة سيقوم المثيل الجديد من التطبيق الرئيسي باستخدام سجل الأعمال لاسترجاع حالة المهام التي تم تنفيذها على التطبيق الذي تعرض للفشل ولا يوجد ضرورة لتنفيذ هذه المهام من جديد. إن عملية الاسترجاع مفعلة بشكل افترضي، ولكن يمكن تعطيلها من خلال إعطاء قيمة false للخاصية yarn.app.MapReduce .am.job.recovery.enable.

يقوم زبون المقابلة والاختزال باستطلاع التطبيق الرئيسي من أجل تقارير التقدم، ولكن إذا تعرض التطبيق الرئيسي الخاص بالزبون للفشل على الزبون أن يقوم بتحديد التطبيق الرئيسي الجديد. خلال مرحلة تهيئة العمل، يقوم الزبون بسؤال مدير الموارد عن عنوان التطبيق الرئيسي، ليقوم بعدها الزبون بحفظ هذا العنوان ليستخدمه عندما يريد استطلاع التطبيق الرئيسي، بهذه الطريقة لن يقوم الزبون

بطلب العنوان مرة ثانية وبالتالي يخفف الحمل على مدير الموارد. إذا تعرض التطبيق الرئيسي للفشل سيقوم الزبون باختبار وقت الانتهاء (timeout) عندما يقوم بإصدار تحديث الحالة، عند هذه النقطة سيقوم الزبون بالرجوع إلى مدير الموارد والسؤال عن عنوان التطبيق الرئيسي الجديد. إن هذه العملية شفافة بالنسبة للمستخدم[2, p. 195].

2-5-9-3 فشل مدير العقدة

إذا فشل مدير العقدة (توقف بشكل مفاجئ أو كان يقوم بالتنفيذ بشكل بطيء جداً)، سيتوقف عن إرسال رسائل نبضات القلب إلى مدير الموارد (أو أنه سيرسل النبضات بشكل غير منتظم). إذا لم يستلم مدير الموارد أي نبضة ولمدة عشر دقائق فإنه سيلاحظ أن مدير العقدة توقف عن إرسال نبضات القلب وسيقوم بإزالته من حوض العقد (nodes pool) التي تتم جدولة الحاويات عليها. كما يمكن تغيير الفترة اللازمة لمدير الموارد لملاحظة فشل مدير العقدة من خلال الخاصية

yarn.resourcemanager.nm.liveness-monitor.expiry-interval-ws وذلك بإعطائها قيمة بالميلي ثانية.

يتم استرجاع المهام التي كانت تنفذ على مدير العقدة الذي تعرض للفشل سواء كانت هذه المهام مهام عادية او أنها تمثل التطبيق الرئيسي، إن عملية الاسترجاع تتم من خلال الآلية التي تم شرحها في الفقرتين السابقتين. كما أن التطبيق الرئيسي يقوم بتنسيق مهام المقابلة –التي تم تشغيلها وانتهت بنجاح على مدير العقدة الذي تعرض للفشل – ليتم تشغيلها من جديد في حال كانت تنتمي لعمل غير مكتمل، وذلك لأن الخرج الوسيط الخاص بهذه المهام يوجد على نظام الملفات الخاص بمدير العقدة الذي تعرض للفشل وبالتالي من المحتمل عدم القدرة على الوصول إليه خلال عملية الاختزال[2, p. 194].

مدراء العقد يمكن أن يتم حظرهم ووضعهم في القائمة السوداء إذا كان عدد المرات التي يفشل فيها التطبيق مرتفع. حتى لو لم يكن مدير العقدة هو الذي فشل بحد ذاته. فإن قائمة الحظر يمكن أن يتم إعدادها من قبل التطبيق الرئيسي، وبالنسبة للمقابلة والاختزال فإن التطبيق الرئيسي سيحاول إعادة جدولة المهام على عقد أخرى في حال فشل أكثر من ثلاث مهام على مدير العقدة. ويمكن للمستخدم أن يقوم بتعديل هذا الحد من خلال الخاصية

MapReduce .job.maxtaskfailures.per.tracker

2-5-9-4 فشل مدير الموارد

إن فشل مدير الموارد يعتبر أمرا خطيرا، لأنه لا يمكن تشغيل المهام ولا الأعمال من دون مدير الموارد، إن مدير الموارد يعتبر نقطة وحيدة للفشل، بالتالي وفي حال فشل ستفشل كل الأعمال ولا يمكن استعادتها.

ومن أجل الوصول إلى توافرية عالية (High availability)، إنه من الضروري تشغيل زوج من مدراء الموارد ليعملا بشكل احتياطي فعال. وفي حال فشل مدير الموارد الفعال، سيقوم المدير الاحتياطي بمتابعة العمل بدون ان يظهر انقطاع واضح للمستخدم.

إن المعلومات المتعلقة بكل التطبيقات التي ما زالت قيد التنفيذ يتم تخزينها في مخزن يتمتع بتوافرية عالية (و هو مدعوم من HDFSأو ZooKeeper[1])، بالتالي سيكون مدير الموارد الاحتياطي قادر على استعادة الحالة المركزية للمدير المركزي الذي تعرض للفشل. إن المعلومات المتعلقة بمدراء العقد لا يتم تخزينها في مخزن الحالة بسبب سهولة تشكيلها نسبيا من قبل مدير الموارد الجديد حالما يقوم مدراء العقد بإرسال نبضات القلب إلى مدير الموارد الجديد، من الجدير ملاحظته أن المهام ليست جزءا من حالة مدير الموارد، وذلك لأنه تتم إدارتها من قبل التطبيق الرئيسي. بالتالي، كمية الحالة الواجب تخزينها أكثر قابلية للإدارة من الحالة الخاصة بالعمل في المقابلة والاختزال.

عندما يبدأ مدير الموارد الجديد يقوم بقراءة معلومات التطبيق من مخزن الحالة، بعدها يقوم بإعادة تشغيل التطبيقات الرئيسية من أجل كل الأعمال التي يتم تنفيذها على العنقود. وهذا لا يعتبر محاولة فاشلة للتطبيق (أي لا يتم احتسابها ضمن yarn.resourcemanager.am.max.attempts) لأن التطبيق لم يفشل بسبب خطأ في الشيفرة البرمجية الخاصة بالتطبيق وإنما تم إنهائه من قبل النظام. عمليا، لا تعتبر عملية إعادة تشغيل التطبيق الرئيسي أمرا مهما في تطبيقات المقابلة والاختزال وذلك لأنه من الممكن لهذه التطبيقات أن تقوم باسترجاع كل الأعمال التي تم تنفيذها من قبل المهام التي انهت عملها بشكل كامل.

[1] هو مخدم مفتوح المصدر ينسق التعاون في النظم الموزعة بشكل موثوق[51]

إن انتقال مدير الموارد من الحالة الاحتياطية إلى الحالة الفعالة يتم من خلال متحكم إنهاء الفشل (failover controller). إن المتحكم الافتراضي لإنهاء الفشل هو متحكم اوتوماتيكي، والذي يستخدم ZooKeeper للتأكد من وجود مدير موارد واحد فعال في نفس الوقت.

يجب أن يتم إعداد الزبائن ومدراء العقد من أجل معالجة إنهاء فشل مدير الموارد، وبسبب وجود مديري موارد يمكن الاتصال بهما. فيقوم الزبائن ومدراء العقد بالاتصال بمدراء الموارد بطريقة راوند روبن (round robin) حتى إيجاد المدير الفعال. و عند فشل المدير الفعال يقوم الزبائن ومدراء العقد بمحاولة الاتصال بنفس الطريقة حتى يصبح المدير الاحتياطي فعالا [2, p. 195].

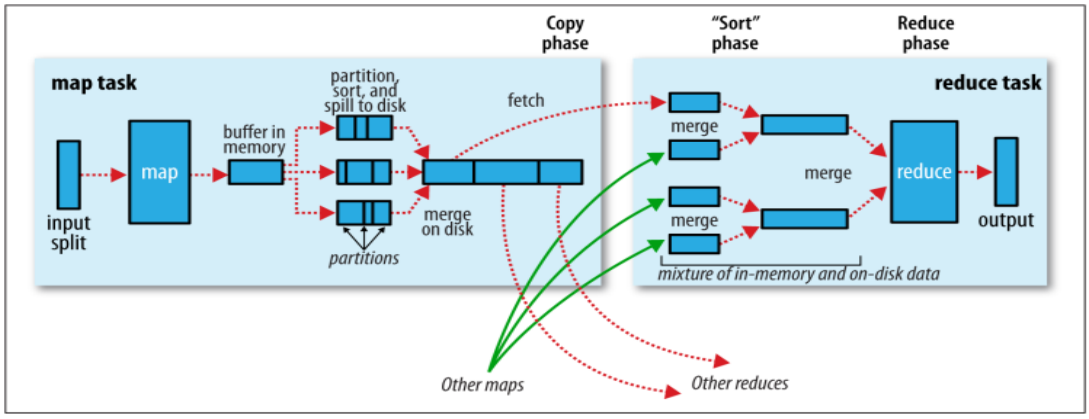
2-5-10 الخلط والفرز (Shuffle and Sort)

إن المقابلة والاختزال تضمن أن دخل كل مختزل يجب أن يكون مرتبا تبعا للمفاتيح. إن العملية التي يقوم بها النظام بإنجاز الترتيب ونقل خرج عملية المقابلة إلى عملية الاختزال تعرف بالخلط (shuffle).

إن عملية الخلط هي جزء الكود الذي يحدث فيه التنقية والتحسين بشكل مستمر، ولا بد من توضيح ذلك بشكل مفصل أكثر، بشكل عام إن عملية الخلط هي قلب المقابلة والاختزال[2, pp. 197–200].

2-5-10-1 من جانب المقابلة

عندما يبدأ تابع المقابلة بإنتاج الخرج، فإن هذا الخرج لن تتم كتابته إلى القرص بكل بساطة. وإنما هنالك العديد من العمليات التي تشمل التخزين في العوازل الذاكرية والقيام ببعض أعمال ما قبل الفرز من أجل تحسين الأداء. إن الشكل (2-15) يبين لنا ماذا يحدث.



الشكل (2-15) الترتيب والخلط في المقابلة والاختزال [2, p. 197]

إن كل عملية مقابلة تمتلك عازل ذاكري دائري تتم كتابة الخرج عليه. إن الحجم الافتراضي لهذا العازل هو 100MB (يمكن تغيير هذا الحجم من خلال الخاصية MapReduce .task.io.sort.mb). عندما تصل محتويات هذا العازل إلى عتبة حجم معينة (MapReduce map.sort.spill.percent. والتي تمتلك القيمة الافتراضية 80% or 0.80)، سيبدأ مسلك يعمل في الخلفية (backgournd thread) بتفريغ المحتوى إلى القرص. ستستمر عملية كتابة خرج المقابلة بينما تتم عملية تفريغ المحتوى إلى القرص، ولكن إذا امتلأ العازل بشكل كامل خلال هذه العملية، ستتوقف عملية المقابلة حتى ينتهي التفريغ بشكل كامل. يتم التفريغ إلى المجلدات المحددة بالخاصية MapReduce .cluster.local.dir وذلك ضمن المجلد الفرعي المحدد والخاص بالعمل.

قبل أن تتم عملية الكتابة للقرص يقوم المسلك المسؤول عن الكتابة بتقسيم البيانات إلى أجزاء متعلقة بعمليات الاختزال التي ستستلم البيانات. من أجل كل قسم يقوم المسلك بإنجاز ترتيب ضمن الذاكرة للبيانات وحسب المفتاح، وفي حال كان هنالك تابع تجميع (combiner function) يتم تطبيق هذا التابع من أجل تقليل خرج عملية المقابلة، بالتالي سيكون حجم البيانات التي سيتم كتابتها على القرص أقل وكذلك الأمر بالنسبة للبيانات التي سيتم إرسالها إلى المختزل عبر الشبكة.

في كل مرة يصل فيها العازل الذاكري إلى عتبة التفريغ، يتم توليد ملف تفريغ جديد، بالتالي وبعد أن تقوم عملية المقابلة بكتابة سجلاتها إلى الخرج، سيكون لدينا عدة ملفات تفريغ (spill files). قبل أن تنتهي المهمة، يتم دمج ملفات التفريغ وتحويلها إلى ملف خرج واحد ومرتب. إن الخاصية MapReduce .task.io.sort.factor تتحكم بالعدد الأعظمي للتدفقات التي يتم دمجها في نفس الوقت وتكون القيمة الافتراضية 10.

اذا كان لدينا ثلاث ملفات تفريغ على الأقل (يمكن تغيير العدد الأصغري لملفات الدمج من خلال الخاصية MapReduce .map.combine.minspills). سيعمل المجمع (combiner) مرة ثانية قبل أن تتم عملية كتابة ملفات الخرج. إن عملية إعادة استدعاء المجمع يمكن أن تتم بشكل متكرر على بيانات الدخل دون أن يؤثر ذلك على النتيجة النهائية. في حال وجود تفريغ واحد أو تفريغين، فإن إمكانية الاختزال في حجم خرج عملية المقابلة مهمل مقارنة بكلفة استدعاء المجمع، بالتالي لن يتم استدعاء المجمع مرة ثانية بالنسبة لخرج المقابلة الحالي.

إن تقليل خرج المقابلة الذي سيتم كتابته إلى القرص يعتبر فكرة سديدة للأسباب التالية:

- تسريع الكتابة على القرص
- يوفر مساحة على القرص
- يقلل كمية البيانات التي سيتم إرسالها إلى المختزل

إن الخرج غير مضغوط بشكل افتراضي، ولكن من السهل السماح بضغط الخرج من خلال إعطاء القيمة true إلى الخاصية MapReduce .map.output.compress. إن المكتبة المستخدمة في الضغط يتم تحديها من خلال MapReduce .map.output.compress.codec.

يتم جعل أجزاء ملفات الخرج متاحة للمختزلات عن طريق البروتوكول HTTP. إن العدد الأعظمي من المسالك العاملة والمستخدمة لتخديم أجزاء الملفات يتم التحكم به من خلال الخاصية MapReduce .shffle.max.threads ; إن هذه الإعدادات يتم ضبطها على مدير العقدة وليس على مهمة المقابلة. والقيمة الافتراضية هي 0 بالتالي سيكون عدد المسالك يساوي ضعفي عدد المعالجات على الآلة [2, pp. 197–199].

2-5-10-2 من جانب الاختزال

لننتقل إلى الجانب الآخر من العملية وهو جانب الاختزال. إن ملفات خرج المقابلة تتواجد على القرص المحلي للآلة التي تقوم بشغيل عمليات المقابلة (من الجدير بالذكر أن خرج المقابلة يتم كتابته بشكل دائم إلى القرص لكن لا يتم كتابة خرج الاختزال إلى القرص بشكل دائم)، إن مهمة الاختزال ولكي تنجز الجزء الخاص بها تحتاج خرج المقابلة الناتج عن العديد من مهام المقابلة في العنقود الحاسوبي. إن عمليات المقابلة يمكن أن تنتهي من التنفيذ في أوقات مختلفة، بالتالي مهام الاختزال تقوم بنسخ الخرج حالما تنتهي كل عملية مقابلة. هذه العملية تعرف بمرحلة النسخ (copy phase) الخاصة بعملية الاختزال. إن مهمة الاختزال تمتلك عدد قليل من مسالك النسخ بالتالي يمكن أن يتم احضار خرج المقابلة بشكل متفرع [2, p. 199].

إن القيمة الافتراضية لعدد مسالك النسخ هي 5 ويمكن التعديل على هذه القيمة من خلال الخاصية reduce.shuffle.parallelcopies

2-5-11 جلب خرج المقابلات.

حالما تنتهي عمليات المقابلة بنجاح، فإن هذه العمليات تقوم بإعلام التطبيق الرئيسي بذلك من خلال آلية نبضات القلب. بالتالي ومن أجل عمل معين يمكن للتطبيق الرئيسي معرفة العلاقة التي تربط بين خرج المقابلة والحواسيب. يقوم المسلك في المختزل وبشكل دوري بسؤال التطبيق الرئيسي عن الأجهزة التي تحتوي خرج المقابلة وتستمر هذه العملية حتى يتم جلب الخرج كاملا.

اذا كان خرج المقابلة صغيرا بشكل كاف فإنه يتم نسخه إلى ذاكرة آلة الجافا الافتراضية الخاصة بمهمة الاختزال (إن حجم العازل يمكن أن يتم التحكم به من خلال الخاصية MapReduce .reduce.shuffle.input.buffer.percent والتي تحدد نسبة من الHeap من أجل ذلك). وفي حال كان خرج المقابلة كبيرا فإنه يتم نسخه على القرص. عندما يصل العازل الذاكري إلى عتبة الحجم (التي يتم التحكم بها من خلال MapReduce .reduce.shuffle.merge.percent) أو أنه وصل إلى العتبة التي تحدد عدد وحدات خرج المقابلة(و التي يتم التحكم بها من خلال MapReduce .reduce.merge.inmen.threshold) يتم دمج الخرج على العازل في الذاكرة وتفريغه على القرص. وفي حال استخدام الضاغط فإنه سيعمل خلال عملية الدمج من أجل تقليل حجم البيانات التي تتم كتابتها على القرص.

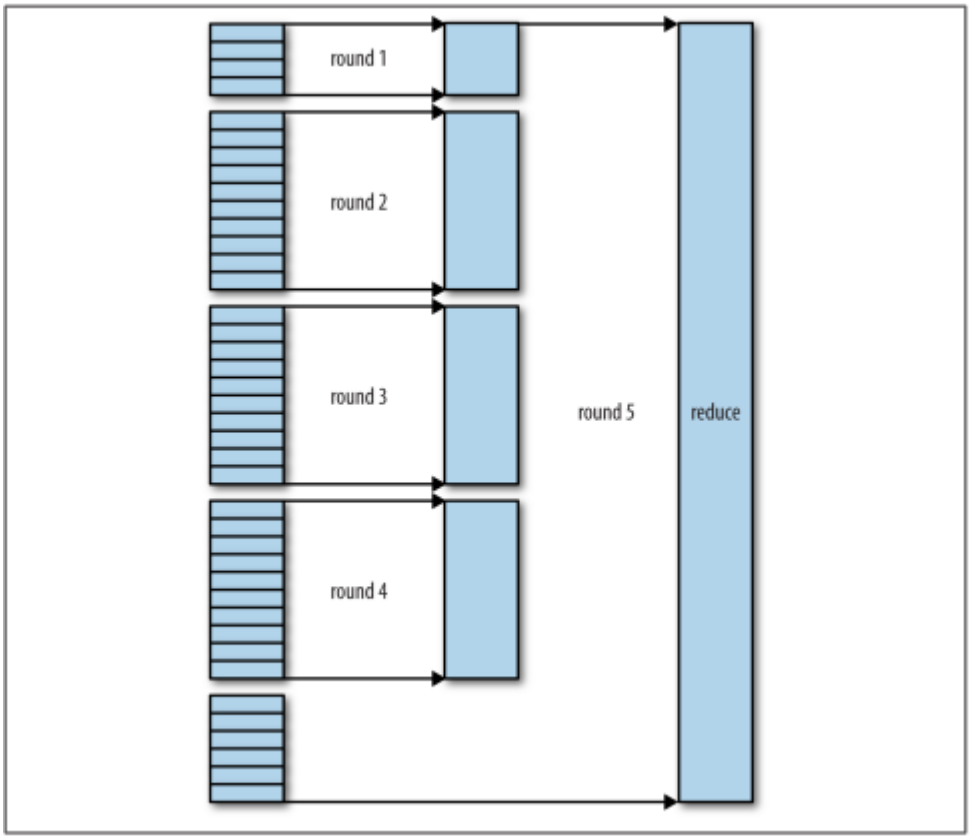
حالما يتم جمع النسخ على القرص، يقوم مسلك يعمل بالخلفية بترتيب ودمج الملفات. إن ذلك يوفر بعض الوقت اللازم للدمج فيما بعد. يجب الانتباه إلى أن أي خرج للمقابلة يتم ضغطه يجب أن يتم فك ضغطه في الذاكرة من أجل الدمج.

عندما يتم الانتهاء من نسخ كل خرج عمليات المقابلة، تنتقل عملية الاختزال إلى مرحلة الترتيب (والتي يجب أن تدعى مرحلة الدمج(merge) لأن عملية الترتيب تم إنجازها أثناء عملية المقابلة)، حيث يتم دمج خرج المقابلة مع مراعات الترتيب. إن ذلك يتم بشكل دوري. على سبيل المثال، وليكن لدينا 50 خرج مقابلة وكان عامل الدمج 10(يمكن التعديل على عامل الدمج من خلال الخاصية MapReduce .task.io.sort.factor كما في حالة الدمج في المقابلة) بالتالي سيكون لدينا خمس دورات للدمج. في كل دورة يتم دمج عشر ملفات إلى ملف واحد، وفي النهاية سنحص على خمسة ملفات وسيطة.

بدل أن يكون لدينا دورة أخيرة لدمج الخمسة ملفات إلى ملف واحد مرتب، فإن عملية الدمج تقوم بتوفير وصول إلى القرص من خلال تغذية تابع الاختزال بما موجود بآخر مرحلة. بالتالي الدمج الأخير يأتي من خلال دمج الأجزاء التي في الذاكرة وعلى القرص.

في الحقيقة إن عدد الملفات التي يتم دمجها في كل دورة غير منتظم وليس كما في المثال التوضيحي. إن الهدف هو دمج أقل عدد ممكن من الملفات للحصول على عامل الدمج في الدورة الأخيرة. ليكن لدينا 40 ملف، إن علمية الدمج لن تقوم بدمج 10 ملفات خلال أربع دورات لنحصل في النهاية على 4 ملفات. إنما سيتم دمج 4 ملفات لنحصل على ملف ومن ثم سيتم دمج عشر ملفات من أجل ثلاث دورات سيتبقى 6 ملفات بالإضافة إلى 3 ملفات هي نتيجة الدمج ليصبح العدد الأخير 10 ملفات يتم دمجهم بالدورة الأخيرة. إن هذه العملية موضحة في الشكل(2–16).

إن ذلك لا يغير عدد موجات الدمج و إنما يساهم بجعل حجم البيانات التي ستتم كتابتها على القرص مثاليا، لأن آخر دورة يتم دمجها بشك مباشر وتوجيهها بعدها إلى الاختزال [2, p. 200].



الشكل (2–16) الدمج الفعال لـ 40 ملف بعامل دمج 10 [2, p. 200]

خلال عملية الاختزال يتم استدعاء تابع الاختزال من أجل كل مفتاح في الخرج المخزن. خرج هذه المرحلة يتم كتابتها إلى نظام ملفات موزع وعادة يكون HDFS والسبب في استخدام HDFS هو أن مدير العقدة يقوم بتشغيل عقدة بيانات (datanode) وأول نسخة من البيانات تتم كتابتها إلى القرص المحلي.

12-5-2 ضبط الإعداد (Configuration tuning)

نحن الآن في أفضل مكان لفهم كيفية ضبط عملية الخلط من أجل تحسين أداء المقابلة والاختزال. إن الإعدادات التي تتعلق بذلك والتي يمكن أن يتم استخدامها من أجل كل عمل (عدا ما ينوه له) مبينة في الجدول (2-2) والجدول (2-3)، كما يحتوي الجدولان على القيم الافتراضية والتي تعتبر جيدة من أجل الأعمال متعددة الأهداف.

إن المفهوم العام يتمثل في إعطاء عملية الخلط أكبر مقدار من الذاكرة. عل أية حال، هنالك معادلة لا بد من القيام بها للتأكد من أن توابع المقابلة والاختزال تحصل على المقدار الكافي من الذاكرة لتستطيع العمل وتنفيذ المطلوب. من هنا يتبين لنا أهمية كتابة توابع المقابلة والاختزال بحيث تستخدم أقل مقدار من الذاكرة–وبشكل أكثر تحديد: يجب على توابع المقابلة والاختزال أن تحتوي على مقدار أعظمي للذاكرة الممكن استخدامها–.

يتم إعطاء الذاكرة إلى آلة الجافا الافتراضية التي يتم تنفيذ توابع المقابلة والاختزال ضمنها، ويتم ضبط ذلك من خلال الخاصية mapred.child.java.opts.property.

بالنسبة لتابع المقابلة، يمكن الوصول إلى أفضل أداء من خلال تجنب حصول أكثر من عملية تفريغ إلى القرص (الحالة المثالية هي عملية تفريغ واحدة). في حال كنا قادرين على توقع خرج عملية المقابلة، يمكننا ضبط الخاصية *.task.io.sort. Map-Reduce بشكل مناسب من أجل تقليل عدد مرات عملية التفريغ. عمليا، علينا زيادة قيمة MapReduce.task.io.sort.mb. هنالك عداد في المقابلة والاختزال يدعى SPILLED_RECORDS يقوم بعد العدد الأعظمي للسجلات التي تم تفريغها إلى القرص أثناء تنفيذ العمل، وهذه المعلومة يمكن أن تساعدنا في عملية الضبط. ومن الجدير بالذكر أن هذا العداد يقوم بعد عمليات التفريغ بالنسبة للمقابلة وبالنسبة للاختزال.

بالنسبة لتابع الاختزال، يمكن الوصول إلى أفضل أداء عندما تبقى البيانات الوسيطة بشكل كامل في الذاكرة. ولكن ذلك لا يمكن أن يتم بشكل افتراضي، لأنه وبشكل عام يتم حجز كامل الذاكرة لتابع

الاختزال. ولكن وفي حال كان تابع الاختزال يتطلب مقدار صغيرا من الذاكرة وإذا قمنا بإعطاء الخاصية MapReduce .reduce.merge.inmem.threshold القيمة 0 والخاصية MapReduce .reduce.input.buffer.percent القيمة 0 يمكن أن يؤدي ذلك إلى تحسن في الأداء.

Property name	Type	Default value	Description
mapreduce.task.io.sort.mb	int	100	The size, in megabytes, of the memory buffer to use while sorting map output.
mapreduce.map.sort.spill.percent	float	0.80	The threshold usage proportion for both the map output memory buffer and the record boundaries index to start the process of spilling to disk.
mapreduce.task.io.sort.factor	int	10	The maximum number of streams to merge at once when sorting files. This property is also used in the reduce. It's fairly common to increase this to 100.
mapreduce.map.combine.minspills	int	3	The minimum number of spill files needed for the combiner to run (if a combiner is specified).
mapreduce.map.output.compress	boolean	false	Whether to compress map outputs.
mapreduce.map.output.compress.codec	Class name	org.apache.hadoop.io.compress.DefaultCodec	The compression codec to use for map outputs.
mapreduce.shuffle.max.threads	int	0	The number of worker threads per node manager for serving the map outputs to reducers. This is a cluster-wide setting and cannot be set by individual jobs. 0 means use the Netty default of twice the number of available processors.

الجدول (2–2) الاعدادات الواجب ضبطها من جهة عملية المقابلة [2, p. 202]

Property name	Type	Default value	Description
mapreduce.reduce.shuffle.parallelcopies	int	5	The number of threads used to copy map outputs to the reducer.
mapreduce.reduce.shuffle.maxfetchfailures	int	10	The number of times a reducer tries to fetch a map output before reporting the error.
mapreduce.task.io.sort.factor	int	10	The maximum number of streams to merge at once when sorting files. This property is also used in the map.
mapreduce.reduce.shuffle.input.buffer.percent	float	0.70	The proportion of total heap size to be allocated to the map outputs buffer during the copy phase of the shuffle.
mapreduce.reduce.shuffle.merge.percent	float	0.66	The threshold usage proportion for the map outputs buffer (defined by mapred.job.shuffle.input.buffer.percent) for starting the process of merging the outputs and spilling to disk.
mapreduce.reduce.merge.inmem.threshold	int	1000	The threshold number of map outputs for starting the process of merging the outputs and spilling to disk. A value of 0 or less means there is no threshold, and the spill behavior is governed solely by mapreduce.reduce.shuffle.merge.percent.
mapreduce.reduce.input.buffer.percent	float	0.0	The proportion of total heap size to be used for retaining map outputs in memory during the reduce. For the reduce phase to begin, the size of map outputs in memory must be no more than this size. By default, all map outputs are merged to disk before the reduce begins, to give the reducers as much memory as possible. However, if your reducers require less memory, this value may be increased to minimize the number of trips to disk.

الجدول (2–3) الاعدادات الواجب ضبطها من جهة عملية الاختزال [2, p. 202]

2–5–13 تنفيذ المهام

تكلمنا حول قيام نظام المقابلة والاختزال بتنفيذ المهام في سياق تنفيذ العمل. سنتكلم في الفقرات التالية في هذا الفصل عن سبل التحكم الواجب على المستخدم القيام بها من أجل التحكم بتنفيذ مهام المقابلة والاختزال[2, p. 203].

2–5–13–1 بيئة تنفيذ المهام

يقوم Hadoop بتزويد مهام المقابلة والاختزال بمعلومات حول البيئة التي يتم تنفيذ هذه المهام عليها. على سبيل المثال، يمكن لمهمة المقابلة أن تستكشف اسم الملف الذي تقوم بمعالجته، كما يمكن لمهمة المقابلة أو مهمة الاختزال أن تحصل على رقم المحاولة الخاصة بالمهمة. إن الخصائص الموضحة

في الجدول (2–4) يمكن الوصول إليها وفي حال الواجهة البرمجية القديمة من خلال تحقيق الدالة configure() ضمن الصف Mapper أو الصف Reducer، حيث يتم تمرير الإعدادات كوسطاء. أما عند استخدام الواجهة البرمجية الجديدة، يمكن الوصول إلى هذه الخاصيات من خلال غرض السياق (context object) والذي يتم تمريره إلى كل الدوال في الصف Mapper أو الصف Reducer.

Property name	Type	Default value	Description
mapreduce.reduce.shuffle.parallelcopies	int	5	The number of threads used to copy map outputs to the reducer.
mapreduce.reduce.shuffle.maxfetchfailures	int	10	The number of times a reducer tries to fetch a map output before reporting the error.
mapreduce.task.io.sort.factor	int	10	The maximum number of streams to merge at once when sorting files. This property is also used in the map.
mapreduce.reduce.shuffle.input.buffer.percent	float	0.70	The proportion of total heap size to be allocated to the map outputs buffer during the copy phase of the shuffle.
mapreduce.reduce.shuffle.merge.percent	float	0.66	The threshold usage proportion for the map outputs buffer (defined by mapred.job.shuffle.input.buffer.percent) for starting the process of merging the outputs and spilling to disk.

الجدول 2–4 اهم الخصائص التي يمكن لمهام المقابلة والاختزال الوصول اليها و الخاصة ببيئة التنفيذ [2, p. 203]

2–5–13–2 التنفيذ التخميني

إن نموذج المقابلة والاختزال يقوم بتقسيم العمل إلى مهام يتم تنفيذ هذه المهام بشكل موزع بحيث يصبح زمن تنفيذ العمل أصغر من الزمن الممكن أن يستهلك أثناء التنفيذ التسلسلي لنفس العمل. إن هذا يبين تأثير المهام التي تعمل بشكل بطيء على زمن التنفيذ، إن وجود مهمة واحد تعمل بشكل بطيء يؤدي إلى إطالة الزمن اللازم لتنفيذ العمل بشكل ملحوظ. لكن العمل يمكن أن يتكون من مئات وربما آلاف من المهام، بالتالي هنالك احتمال لوجود بعض المهام البطيئة.

هنالك عدة أسباب قد تؤدي إلى جعل المهمة بطيئة:

- إعدادات برمجية خاطئة

- انخفاض في العتاد الصلب

ولكن يمكن أن يصعب اكتشاف السبب لكون المهمة ما زالت تتابع التنفيذ بشكل صحيح، على الرغم من استهلاكها زمن أكبر من المتوقع. إن Hadoop لا يحاول أن يكتشف ويصلح المهام التي تنفذ بشكل بطيء، وبدلا من ذلك يقوم Hadoop بمحاولة اكتشاف الحالات التي تكون فيها المهمة تقوم بالتنفيذ بشكل أبطأ من المتوقع ليقوم عندها Hadoop بتشغيل مهمة جديدة كمهمة احتياطية. وهذا يدعى التنفيذ التخميني للمهام.

من الضروري أن ننتبه إلى أن التنفيذ التخميني لا يعمل بتشغيل مهمتين متطابقتين في نفس الوقت وكأننا نسمح بالتسابق بينهما. إن هذا الخيار يؤدي إلى هدر في موارد العنقود. كما أن المجدول يقوم بتتبع تقدم كل المهام في العمل التي تنتمي لنفس النوع (مقابلة واختزال) ، ولا يقوم بتشغيل نسخ تخمينية إلا للمهام التي تقوم بتنفيذ بشكل أبطء بكثير من المعدل العام. عندما تنتهي المهمة بشكل صحيح، يتم إنهاء المهمة المطابقة لأنه لم يعد ضرورة لعملها. بالتالي عندما تنتهي أي من المهمتين (المهمة الأصلية أو المهمة التخمينية) يتم إنهاء المهمة الأخرى.

إن التنفيذ التخميني يعتبر تحسينا ولا يعتبر ميزة تجعل من العمل يعمل بشكل أكثر وثوقيه. ففي حال وجود أخطاء برمجية تؤدي إلى توقف المهام أو تنفيذها بشكل بطيء يعتبر اللجوء إلى التنفيذ التخميني خيارا خاطئ ولن يساعد في جعل العمل أكثر وثوقيه، وذلك لأن الأخطاء البرمجية التي أثرت على تنفيذ المهمة الأساسية ستؤثر على المهمة التخمينية. بالتالي لا بد من إصلاح الخطأ البرمجي لتجاوز المشاكل التي تندرج تحت هذا النوع [2, p. 205].

يتم تشغيل التنفيذ التخميني بشكل افتراضي. ويمكن إطفاءه وتشغيله بشكل مستقل بالنسبة لمهام المقابلة والاختزال، ويمن التحكم بذلك على مستوى العنقود وعلى مستوى العمل والجدول (2–5) التالي يبين أهم الخاصيات التي تتعلق بالتنفيذ التخميني.

Property name	Type	Default value	Description
mapreduce.map.speculative	boolean	true	Whether extra instances of map tasks may be launched if a task is making slow progress
mapreduce.reduce.speculative	boolean	true	Whether extra instances of reduce tasks may be launched if a task is making slow progress
yarn.app.mapreduce.am.job.speculator.class	Class	org.apache.hadoop.mapreduce.v2.app.speculate.DefaultSpeculator	The Speculator class implementing the speculative execution policy (MapReduce 2 only)
yarn.app.mapreduce.am.job.task.estimator.class	Class	org.apache.hadoop.mapreduce.v2.app.speculate.LegacyTaskRuntimeEstimator	An implementation of TaskRuntimeEstimator used by Speculator instances that provides estimates for task runtimes (MapReduce 2 only)

الجدول (2–5) الخاصيات التي تتعلق بالتنفيذ التخميني [2, p. 205].

قد نتساءل عن السبب الذي قد يدفعنا إلى إيقاف التنفيذ التخميني. إن الهدف من التنفيذ التخميني يكمن في تقليل الزمن اللازم لتنفيذ العمل، لكن ذلك يؤثر على أداء العنقود. فمن أجل العناقيد المزدحمة، يمكن للتنفيذ التخميني أن يقلل إنتاجية العنقود، بسبب المهام الإضافية المستخدمة لتقليل زمن التنفيذ بالنسبة لعمل واحد. ولهذا السبب يقوم بعض مدراء العناقيد بإيقاف التنفيذ التخميني على عناقيدهم ويسمحون للمستخدمين بتشغيلها حسب الرغبة بالنسبة للأعمال الخاصة بهم.

وفي بعض الحالات يمكن أن يكون إيقاف التنفيذ التخميني أمرا جيدا بالنسبة لمهام الاختزال، لأن أي مهمة اختزال مكررة ستقوم بجلب نفس البيانات من خرج عمليات المقابلة، وهذا قد يؤدي إلى زيادة الازدحام في الشبكة.

6-2 الخاتمة

إن إطار المقابلة والاختزال هو إطار موزع لمعالجة البيانات الضخمة المخزنة على نظم ملفات موزعة بالتالي تحدثنا في هذا الفصل عن HDFS وهو نظام الملفات الموزع الخاص بـ Hadoop من حيث اسلوب التكرار والكتابة والقراءة. بعدها تكلمنا عن اسلوب حجز الموارد وجدولة الأعمال حيث تحدثنا بشيء من التفصيل عن YARN وهو مدير الموارد المستخدم في Hadoop. تكلمنا في القسم الأخير من هذا الفصل عن إطار المقابلة والاختزال بشكل تفصيلي من حيث اسلوب عمل المقابل والمختزل والدامج والمجمع بحيث نمهد للفصول الرابع (تصميم وتحقيق المحاكي) حيث سنقوم بتحقيق معظم هذا التفاصيل.

الفصل الثالث

الدراسة المرجعية

3–1 المقدمة

يعتبر نموذج المقابلة والاختزال نموذجا جديدا نسبيا كما أن MRPerf [12] وMumak [13] هما أول محاكيان لهذا النموذج. سنتناول في هذا الفصل الجهود المبذولة في مجال محاكاة المقابلة والاختزال وسنختم هذا الفصل بجدول يقارن بين أهم المحاكيات من حيث أسلوب التحقيق والميزات التي يدعمها كل محاكي.

3–2 محدودية محاكيات الحوسبة الشبكية

هنالك تشابه بين نموذج المقابلة والاختزال والحوسبة الشبكية (Grid Computing[1]) وقد تم تطوير العديد من المحاكيات بهدف محاكاة أنظمة الحوسبة الشبكية وأهما:

- MicroGrid [14]
- SimGrid [15]
- GridSim [16]

المحاكي MicroGrid يعتبر Emulator وليس Simulator. إن النتائج التي يعطيها هذا المحاكي هي نتائج دقيقة جدا إلا أنه يستهلك الوقت بشكل كبير لأنه يعمل على تشغيل نسخ افتراضية من المصادر. إن التطبيقات التي تتم نمذجتها باستخدام MicroGrid يجب أن يتم تحقيقها كما لو انه سيتم تنفيذها على بيئة حقيقية. بالتالي تطوير نموذج ليعمل على MicroGrid هو أمر يتطلب جهد أكبر من تطوير نماذج على باقي المحاكيات.

المحاكي SimGrid: إن SimGrid هو إطار من أجل تطوير محاكيات للتطبيقات الموزعة التي يتم تنفيذها على بيئات موزعة، بحيث تستخدم هذه المحاكيات من أجل تقييم ومقارنة إعدادات البيئات وتصميم النظم والخوارزميات. تعتبر SimGrid متعددة الاستخدام لأنها تؤمن نماذج وواجهات برمجية

[1] الحوسبة الشبكية هي الحوسبة التي تساعد على حل المسائل الضخمة باستخدام بنية حاسوبية موزعة.

APIs من أجل محاكات النظم الموزعة الشهيرة (الشبكات المحلية والشبكات الواسعة والعناقيد الحاسوبية ومراكز البيانات)

ميزات SimGrid:

1- الدقة: تم التحقق من دقة SimGrid بشكل نظري وعملي.
2- قابلية التوسع: إن نموذج المحاكاة في SimGrid و المحاكيات المبنية بالاعتماد عليها تعتبر سريعة و غير مستهلكة للذاكرة بشكل كبير بحيث يمكن تشغيلها بشكل سريع على عقدة واحدة.
3- الاستخدام: إن SimGrid برمجية مفتوحة المصدر و مجانية و يمكن تشغيلها على نظم التشغيل (Windows , Max OS and Linux) و يمكن للمستخدم كتابة المحاكيات الخاصة به باستخدام عدة لغات برمجة (C++,Python or Java).

المحاكي GridSim مشابه للمحاكي السابق إلا أنه محقق باستخدام لغة Java كما أنه يعتمد في بنيته التحتية على مكتبة المحاكاة SimJava [17] والتي تعتمد على مبدأ محاكاة الأحداث المتقطعة.

لا يمكن لمحاكيات الحوسبة الشبكية أمثال (MicroGrid, SimGrid, GridSim) محاكاة إطار المقابلة والاختزال بشكل صحيح. لأن هذه المحاكيات تقوم بنمذجة الأعمال المرسلة للنظام كأعمال دفعية (batch jobs) حيث ان كل عمل يشتمل على كلفة حسابية معينة ويحتوي على توصيفات مسبقة للدخل وللخرج. لكن التفاعلات –في إطار المقابلة والاختزال–بين القرص الصلب والمعالج وبطاقة الشبكة أكثر تعقيدا ولا يمكن اعتبارها عمل دفعي لأن ذلك سيسبب خسارة كبيرة في الدقة. على سبيل المثال فإن مرحلة الاختزال مرتبطة بشكل كبير وتتأثر بسلوك كل الأعمال التي تم تنفيذها في مرحلة المقابلة ولا يوجد أي نظام دفعي يمكنه محاكاة هذه التفاعلات بشكل دقيق. بالتالي لا بد من محاكيات مخصصة لتطبيقات المقابلة والاختزال و هذه المحاكيات قد تستخدم في بنيتها التحتية محاك شبكي مثل MRSim [18]الذي يستخدم GridSim [16] وهذا ما سنتناوله في الفقرات التالية.

3-3 محاكيات المقابلة والاختزال

قام الباحث في المرجع [19] بتقديم المحاكي MRTune من أجل توقع زمن تشغيل أعمال المقابلة والاختزال. ويعتبر هذا المحاكي أول محاكي يأخذ بالحسبان انحراف البيانات(Data Skew) [1] وفشل المهام. إن هذا المحاكي يراعي توزع البيانات ويمكنه أن يعمل بشكل جيد عند انحراف البيانات كما يمكنه أن يتوقع زمن التنفيذ حتى ولو كان فشل المهام أمر يصعب توقعه. تم تقييم هذا المحاكي من خلال بيانات دخل تتبع توزيع Zipfian[20]. بينت النتائج أن MRtune يستطيع أن يتوقع زمن التنفيذ لأعمال المقابلة والاختزال بشكل دقيق حتى لو تعرضت البيانات للانحراف. حيث تم تجريب هذا المحاكي من أجل عدد من التطبيقات (عداد الكلمات والفهرسة المعكوسة) وتمت مقارنة هذه النتائج مع التجربة الواقعية على عنقود Hadoop مؤلف من عقدة رئيسية وخمس عقد ثانوية. إلا أن هذا المحاكي لم يقم بمقارنة هذه النتائج مع نتائج محاك آخر كما أنه لم يوضح الآلية المستخدمة للتحقيق.

MRPerf : اقترح Guanying Wang و آخرون المحاكي MRPerf [12] والذي تم تحقيقه باستخدام C++ و Python و TCL. قام الباحثون بتقديم تصميم دقيق لهذا المحاكي بغية تسهيل تصميم تطبيقات المقابلة والاختزال. يقوم هذا المحاكي بالتقاط مفاهيم عديدة لإعداد المقابلة والاختزال(زمن تنفيذ المقابلات و المختزلات و الحجوم المكتوبة على القرص و الذاكرة)، ويقوم باستخدام هذه المعلومات لتوقع أداء التطبيق. قام الباحثون بتصميم MRPerf كأداة تعمل كبنية تحتية للمقابلة والاختزال، ولتكون هذه الأداة بمثابة أداة تخطيط تجعل من عملية تطوير تطبيقات المقابلة والاختزال أمرا أكثر سهولة من خلال تقليل عدد المتحولات التي يتم إعدادها بشكل يدوي وبشكل تجريبي.

قام الباحثون باختبار عمل المحاكي باستخدام بيانات تم الحصول عليها من عنقود حاسوبي متوسط الحجم. بينت نتائج الاختبارات أن هذا المحاكي قادر على توقع أداء التطبيق بشكل دقيق.

---

[1] انحراف البيانات يعني عدم التوازن في كمية البيانات المخصصة لكل مهمة أو عدم التوازن في حجم العمل المطلوب لمعالجة البيانات،أما في اطار المقابلة و الاختزال فإن انحراف البيانات يحدث عندما تحتوي إحدى العقد على بيانات مُعيَّنة لمعالجتها أكثر من العقد الأخرى إما في مرحلة المقابلة أو مرحلة الاختزال [52]

Overview	Actual	*MRPerf*
Number of map tasks	480	476
Number of reduce tasks	16	16
Total input data	32G	32G
Total output data	32G	32G

Phases	Actual	*MRPerf*
Map	220.0	220.8
Shuffle	7.4	5.4
Sort	0.5	3.4
Reduce	137.9	135.9

Map break-down	Actual	*MRPerf*
map	2.14	2.10
sort	1.12	1.19
spill	4.22	4.58
merge	4.52	4.26
overhead	1.79	1.61
sum	13.80	13.75

Data locality	Actual		*MRPerf*	
	num	time	num	time
Data-local	468	13.77	468	13.66
Rack-local	6	13.60	3	14.67
Rack-remote	6	16.10	5	21.64

الشكل (3–1) أداء تطبيق TeraSort على المحاكي MRPerf [12]

و بالرجوع إلى النتائج التجريبية [21][12] والمبينة بشكل مختصر في الشكل (3–1) فإن MRPerf أعطى دقة عالية في المحاكاة على الرغم من تغير هيكلية الشبكة. إن هذه الدقة تعود لسببين هما:

- إن هذا المحاكي يعتمد على المحاكي الشبكي NS2 من أجل نمذجة العناصر الشبكية. إن المحاكي الشبكي NS2 تم تقديمه منذ سنوات وقت تم اثبات قدرته على دعم محاكاة بروتوكولات النقل (.. TCP) وبروتوكولات التوجيه والـ MULTICAST بالنسبة للشبكات السلكية واللاسلكية بالتالي سيتمكن MRPerf من إعطاء دقة عالية عند محاكاة سلوك الشبكة [22].
- MRPerf يشتمل على العديد من التطبيقات الشهيرة مثل TeraSort والبحث والفهرسة، والتي تعتبر من التطبيقات المعيارية للمقابلة والاختزال ونتائجها مقنعة جدا.

YARNSim: قدم الباحثون N.Liu and et al [23] المحاكي YARNSim و هو أول محاكي يوافق نسخة Haddop 2 والتي تحتوي على YARN، تم تحقيق هذا المحاكي بلغة C بالاعتماد على مكاتب المحاكاة التالية:

- Rensselaer Optimistic Simulation System (ROSS) [24] هذه المكتبة عبارة عن محاك موزع مكتوب بلغة C ويدعم خوارزميات التزامن المحافظة والمتفائلة.
- Co–Design of Exascale Storage System (CODES) [25] هو عبارة عن محاكي يعتمد على ROSS ويهدف لمحاكاة أنظمة التخزين بشكل دقيق ومفصل

إن YARNSim يعتمد نموذج ثابت لنمذجة أداء الذاكرة والمعالج وفق المعادلات التالية

$$T_{\mathrm{m}} = \frac{L_{\mathrm{data}}}{B_{\mathrm{m}}} + t_{\mathrm{m}} \qquad 1\text{-}3$$

$$T_{\mathrm{cpu}} = \alpha_{\mathrm{a}} \cdot \frac{L_{\mathrm{data}}}{P_{\mathrm{cpu}}} + t_{\mathrm{cpu}} \qquad 2\text{-}3$$

حيث:

- T_M زمن القراءة أو الكتابة من الذاكرة
- T_{CPU} زمن المعالجة على المعالج
- L_{DATA} حجم البيانات
- A_a كثافة البيانات وهذا المتحول يختلف باختلاف التطبيق
- P_{cpu} سرعة المعالج
- B_m سرعة النقل مع الذاكرة
- t_m تأخير النقل مع الذاكرة
- t_{cpu} تأخير المعالجة

و يعتمد YARNSim على المحاكي CODES لنمذجة هيكلية الشبكة والقرص الصلب. وعند الرجوع إلى الكود المصدري المسؤول عن محاكاة القرص الصلب تبين أن النموذج يعمل على عدد من المعاملات التي تمثل سرعة القراءة من القرص الصلب وسرعة الكتابة وزمن الوصول (Seek time) وبالاضافة إلى Overhead (الزمن اللازم للقرص الصلب لإصدار أمر القراءة أو الكتابة) وهذه المعاملات تختلف باختلاف حجم البيانات المراد كتابته أو قراءته.

بينت النتائج التجريبية أن زمن التنفيذ المتوقع والناتج عن المحاكي قريب إلى زمن التنفيذ الفعلي على عنقود Hadoop مع نسبة خطأ لا تتجاوز 10% وذلك من أجل عدد من التطبيقات (Terasort, Wordcount)

إلا أن هذا المحاكي لم يتطرق إلى موضوع تحمل الفشل في المقابلة والاختزال كما أنه لم يدعم إلا نمط واحد لجدولة الأعمال (FIFO). كما أن النموذج الرياضي المعتمد لنمذجة المعالج لن يعطي نتائج دقيقة في حال وجود عدد كبير من المهام المنتظرة وذلك بسبب عدم القدرة على حساب التأخير في معالجة المهام.

MRSG : قم الباحثون W.Kolberg and et al المحاكي MRSG [26] و الذي تم بناؤه بالاعتماد على إطار SimGrid[15] (إطار مستخدم لتقييم خوارزميات العناقيد الحاسوبية والGrid وشبكات الند للند) تم تجريب هذا المحاكي على Grid'5000 (French scientific grid) بينت النتائج التجريبية أن أداء هذا المحاكي جيد من حيث السرعة لأنه لا يهتم بالتفاصيل ضمن العقدة فهو يعتبر أن كل عملية لديها كلفة بحيث تكون هذه الكلفة اجمالية دون التفصيل في زمن المعالجة وزمن الوصول للقرص الصلب. إن هذه الآلية تزيد من سرعة المحاكي وتقلل من تعقيد التحقيق الا ان ذلك لا يعطي انطباع صحيح عن أزمنة التنفيذ و استهلاك الموارد في حال التنافس بين الأعمال.

MRSim: قدم S.Hammoud and et al المحاكي MRSim [18] تطرق الباحثون إلى نمذجة الوحدات الأساسية لHadoop آخذين بعين الاعتبار الوحدات الثانوية وذلك بغية الوصول إلى أفضل دقة للنتائج. فعلى سبيل المثال إن كلفة عملية مقابلة واحدة هي ناتجة عن عدد من التفاعلات بين الذاكرة والمعالج والقرص الصلب ووصلة الشبكة ونفس الشيء بالنسبة لعملية الاختزال. إن هذا التصميم أدى إلى الوصول إلى نتائج دقيقة من حيث زمن التنفيذ ومتوسط زمن عملية المقابلة ومتوسط زمن عملية الاختزال. تم بناء هذا المحاكي بالاعتماد على مكتبة المحاكاة SimJava [17] وعلى المحاكي الشبكي GridSim [16]. يمكن لهذا المحاكي أن يقوم بمحاكاة عنقود حاسوبي يحتوي عقد غير متجانسة وذلك بسبب القدرة على نمذجة عقد مختلفة من حيث سرعة المعالج وعدد النوى وسرعة القراءة والكتابة على القرص الصلب. كما اعتماد صيغة JSON تسهل على المستخدم إعداد التجربة. لكن هذا المحاكي يعاني من محدودية في التوسعية و ذلك بسبب مكتبة GridSim و SimJava حيث يتم استخدام عدد كبير من المسالك و من اجل كل كيان محاكاة[16].

HSim : قدم Y.liu و آخرون المحاكي HSim [21] والذي يقوم بنمذجة عدد كبير من المتحولات التي تؤثر على سلوك عقد المقابلة والاختزال، إن هذه المتحولات مقسمة على الشكل التالي:

1– متحولات العقدة

- المعالج: إن HSim يدعم معالج واحد لكل عقدة ولكن يمكن لهذا المعالج أن يحتوي على أكثر من نواة ومن الممكن تعريف سرعة المعالج.
- القرص الصلب: تبعا لتجارب أجراها الباحثون على قرص صلب من النوع Seagate Barracuda 1 TB وجدوا أن سرعة الكتابة العظمى على هذا القرص هي 60MBps وسرعة القراءة العظمى 100MBps وأن أقل سرعة كتابة هي 25MBps وأقل سرعة قراءة هي 55MBps. كما قدموا المعامل r وهو مقدار فاقد السرعة (قراءة أو كتابة) من أجل كل ثانية وإن قيمة هذا المعامل تساوي 0.0056وذلك تبعا للاختبارات التي أجراها الباحثون. وخرج الباحثون بالمعادلة التالية التي تعطينا سرعة الكتابة أو القراءة.

$$x = \frac{x_{\min} x_{\max}}{(x_{\min} - x_{\max})e^{-rt} + x_{\max}}. \qquad 3\text{-}3$$

- الذاكرة: من أجل كل عنصر ذاكري فقد تم نمذجة متحولين فقط وهما سرعة القراءة وسرعة الكتابة على الذاكرة.
- بطاقة الشبكة: من أجل كل بطاقة شبكية تم نمذجة سرعة الإرسال وسرعة الاستقبال.

2– متحولات العنقود: إن متحولات العنقود تعبر عن تفصيلات عنقود Hadoop المراد محاكاته، ويمكن تقسيمها إلى.

- عدد العقد: بحيث يتراوح هذا العدد بين 1 إلى بضع مئات من العقد.
- الهيكلية: إن عدد العقد يمكن أن يتم تنظيمه بهيكلية شبكية معينة. لكن Hsim لا يدعم إلى شبكة برف واحد (Single Rack).
- أدوات الشبكة: سرعة المبدلات الشبكية.
- رتل الأعمال ومجدول الأعمال: إن رتل الأعمال يحتوي على الأعمال التي تنتظر أن يتم تنفيذها. وتبعا لمجدول الأعمال فإن الأعمال تنتظر لفترة معينة من أجل الحصول على الموارد. إن Hsim يحتوي على نوعين للجدولة هما: 1– الداخل أولا خارج أولا 2– fair scheduler وهو بذلك يتغلب على YARNSim [23] الذي لا يدعم إلى نمط الداخل أولا خارج أولا.

3– متحولات نظام Hadoop (Hadoop system parameters)

قبل أن يبدأ تطبيق Hadoop بمعالجة البيانات، يتوجب نسخ البيانات إلى نظام الملفات الموزع (HDFS)، وعادة ما يكون عدد نسخ المقابلة (Map instances) مساويا لعدد أجزاء البيانات (chunks) وفي حال كان عدد أجزاء البيانات أكبر من عدد نسخ المقابلة بالتالي يتم اسناد البيانات إلى نسخ المقابلة على شكل دفعات. هنالك العديد من المتحولات التي تتحكم بنظام Hadoop ككل وتشتمل على:

- توصيفات الأعمال:
  - معرف العمل
  - حجم العمل ككل بغض النظر عن عدد أجزاء البيانات
  - عدد السجلات: يساعد هذا المتغير على حساب حجم السجل بالاعتماد على الحجم الكلي للسجلات، كم أن هذا العدد يستخدم عمليا لحساب عدد السجلات المراد دمجها ويعتبر عاملا مهم في الأداء.
  - نسبة خرج المقابلة (Map output ratio) يحدد هذا المعامل حجم البيانات الوسيطة الناتجة عن نُسخ المقابلة.
  - نسبة خرج الاختزال (Reduce output ratio) وهو مشابه للمعامل السابق
  - NumberOfChunks يحدد هذا المتغير عدد الملفات التي تحتوي على البيانات
  - NumberOfReducers عدد نسخ الاختزال اللازمة للعمل.
- المتغيرات الخاصة بمحاكاة Hadoop
  - io.sort.mb حجم العازل الذاكري المستخدم من أجل ترتيب خرج المقابلة.
  - io.sort.spill.percent يحدد العتبة التي يبدأ من بعدها المقابل بتفريغ البيانات من الذاكرة إلى القرص الصلب.
  - io.sort.factor (1) يحدد عدد المسالك الأعظمي والتي سيتم دمجها في عملية المقابلة.
  - io.sort.factor (2) يحدد عدد المسالك الأعظمي والتي سيتم دمجها في عملية الاختزال.

4– المتحولات الخاصة بـ HSim: هنالك خمس متحولات أساسية تتحكم بسلوك HSim وهي:

- ساعة النظام: عبارة عن مكون مستمر من الناحية الزمنية. فمن أجل أي تعديل على ساعة النظام، يتم إضافة ثانية للقيمة الحالية.
- سرعة التنفيذ: يحدد سرعة التنفيذ لكل مكونات HSim
- مستوى الدقة: من أجل معظم تطبيقات Hadoop يكون مستوى الدقة هو الثواني ولكن يمكن من أجل الحصول على دقة أفضل أن نستخدم الميلي ثانية.
- المتحولات المشتركة: تحدد هذه المتحولات نسبة الموارد المشتركة مثل القرص الصلب وعرض الحزمة. وتعرف النسبة بالشكل التالي.

  r = AssignedResource/TotalResource
- Reporter يقوم بحفظ حالة النظام في تقارير من أجل عمليات التحليل.

و الجدول(3–1) يبين جميع المتحولات التي تم ذكرها.

قام الباحثون بعدد كبير من التجارب لاختبار الأداء وتبين من خلالها قدرة هذا المحاكي على محاكاة السلوك الديناميكي لHadoop .

The parameters modeled in HSim.

Category	Specification
Node parameters	Processor, hard disk, memory, Ethernet card, Map instance, Reduce instance
Cluster parameters	Number of nodes, topology, network facilities, job queue, job scheduler
Hadoop system parameters	Job specifications, Hadoop parameters
HSim parameters	System clock, execution speed, accuracy level, shared parameters, reporter

الجدول (3–1) المتحولات التي تم استخدامها في HSim [21]

يعاني هذا المحاكي من المحدوديات التالية:

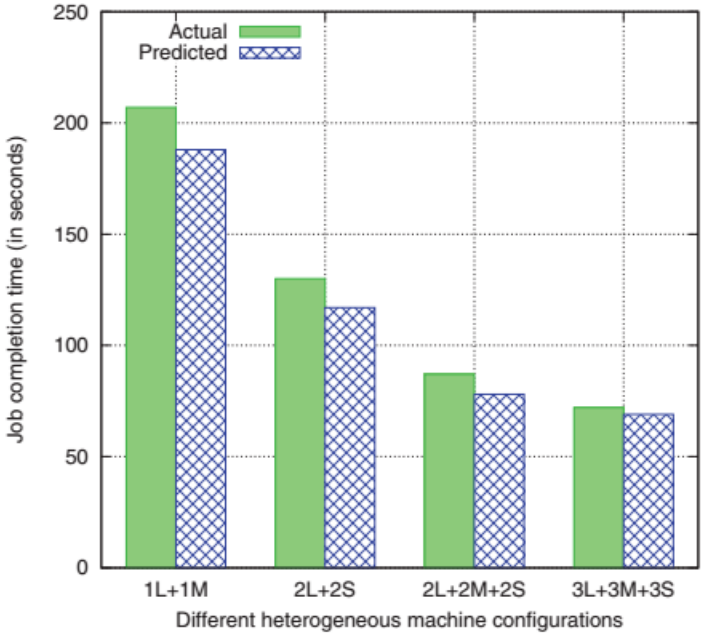
- لا يدعم نمط الجدولة capacity
- لا يدعم إلا رف واحد من حيث هيكلية الشبكة
- كما لم يتم الأخذ بعين الاعتبار موضوع تحمل الفشل.

Mumak : محاك آخر هو Mumak [13] وهو عبارة عن مشروع مفتوح المصدر يؤمن أداة للباحثين والمطورين من أجل نمذجة الميزات (مثل جدولة المقابلة والاختزال) ومن أجل توقع سلوك وأداء هذه الميزات وذلك من أجل مقدار معقول من الموثوقية. يقوم Mumak بأخذ أثر العمل (job trace) الذي تم تنفيذه على عنقود حاسوبي حقيقي. بعدها يقوم Mumak بمحاكاة هذه الأعمال على عنقود حاسوبي افتراضي. سيكون الخرج عبارة عن أثر مفصل لتنفيذ العمل ومسجل تبعا لوقت المحاكاة الافتراضي. إن تحليل الخرج يمكن أن يساعد على فهم أعمق لتأثير استخدام خوارزميات جدولة مختلفة. إن Mumak لا يحاكي الموارد التي يتم مشاركتها زمنيا في المستويات الأدنى. فهذا المحاكي لا يحاكي مهام المقابلة والاختزال بشكل فعلي وإنما يقوم بأخذ السجلات الناتجة عن تنفيذ الأعمال على العناقيد الحاسوبية الواقعية وبعد ذلك يقوم بإسناد هذه المهام تبعا لخوارزمية جدولة مختلفة أو تبعا لعنقود جديد. ولكن المهام وبشكل مؤكد سوف تستهلك وقت تنفيذ مختلف عند استخدام سياسة جدولة مختلفة. وبالتالي ستكون نتائج التوقع غير واقعية عند محاكاة العمل على عناقيد مختلفة أو على نفس العنقود ولكن من أجل إعدادات مختلفة. وإن ذلك يعتبر المحدودية الرئيسية التي يعاني منها Mumak. ليكن لدينا المثال التالي بحيث نوضح الخطأ الحاصل في التوقع عند استخدام Mumak: فعندما نقوم بمضاعفة عدد المهام الممكن تنفيذها على العقدة الواحدة في نفس الوقت، سيستمر Mumak بتوقع نفس الزمن لكل مهمة وبالتالي سيتناقص الوقت اللازم لتنفيذ العمل إلى النصف. لكن ومن أجل عنقود حقيقي، فإن هذا السيناريو سيقوم بشكل تقريبي بمضاعفة زمن التنفيذ لكل مهمة وذلك بسبب تضاعف عدد العمليات التي تتشارك نفس العتاد الصلب (القرص الصلب، الشبكة، المعالج) ولكن الوقت الإجمالي للعمل لن يتغير بشكل كبير.

MR: R.Singhal and A.Verma [27] قاما بتطوير المحاكي، تعتمد فكرة هذا المحاكي على جمع قياسات لمهام المقابلة والاختزال من خلال تنفيذ التطبيق على عنقود غير متجانس ومن أجل حجوم بيانات صغيرة. بعدها قام الباحثين باستخدام علاقة خطية من أجل توقع زمن التنفيذ لمهام المقابلة والاختزال من أجل حجوم بيانات أضخم ومن أجل عناقيد حاسوبية تحتوي على موارد مختلفة. وبشكل أكثر تفصيلا، فقد قام الباحثان بإعداد عنقود Hadoop غير متجانس بحيث تكون كل عقدة في هذا العنقود تمتلك عتاد صلب مختلف. فعلى سبيل المثال وإذا كان العنقود يحتوي على 16 عملية مقابلة وحجم جزء البيانات في نظام الملفات الموزع 64MB سيكون حجم بيانات الدخل 16*64MB=1GB ومع ازدياد حجم بيانات الدخل سيزداد عدد عمليات المقابلة بحيث يكون حجم

جزء البيانات ثابت. بالتالي سيكون زمن عملية المقابلة ثابت. وفي المقلب الآخر وإذا كان عدد عمليات الاختزال ثابتا مع ازدياد حجم بيانات الدخل بالتالي سيزداد حجم بيانات دخل عمليات الاختزال وهذا يعني زيادة في زمن عمليات الاختزال بشكل خطي. وعلى سبيل التبسيط افترض الباحثان أن البيانات تنمو بشكل منتظم وتتوزع على عمليات الاختزال بشكل منتظم.

تم اختبار هذا المحاكي على عدد من التطبيقات ومن أجل عناقيد تحتوي على بنى صلبة غير متجانسة وبينت الاختبارات أن نسبة الخطأ في توقع زمن التنفيذ لا تتعدا 10%. والشكل (3–2) يبين ذلك.



الشكل (3–2) مقارنة أداء المحاكي MR مع التجربة الواقعية و من أجل عناقيد غير متجانسة [27]

حيث:

L أي أن العقدة كبيرة Large وتحتوي على معالج بثمان نوى

M أي ان العقدة متوسطة Medium وتحتوي على معالج بأربع نوى

S أي ان العقدة صغيرة Small وتحتوي على معالج بنواتين

والأرقام التي تسبق هذه الأحرف تعبر عن عدد العقد من هذا النوع والموجودة ضمن كل عنقود. ما يميز هذا المحاكي هو السرعة في حساب الزمن المتوقع لإنجاز أعمال المقابلة والاختزال لكنه لا

يراعي موضوع الجدولة عند تنفيذ أكثر من مهمة كما أنه لا يعطي تصور كامل لحالة كل عقدة من حيث استخدامية المعالج وكرت الشبكة والقرص الصلب.

WaxElephant : قام Zujie Ren وآخرون بتقديم المحاكي WaxElephant [28] وهو محاكي أحداث متقطعة يدعم الإصدار 1 من Hadoop ويحتوي على ألية لمحاكاة القرص الصلب والمعالج ونظام الملفات الموزع كما يدعم التكرار في عملية الكتابة. أما بالنسبة لأنماط الجدولة فهذا المحاكي يدعم الجدولة من نوع FIFO وFair وCapacity، إلا انه لا يدعم العناقيد الحاسوبية إلا من أجل رف واحد بالتالي هنالك محدودية بالتوسعية، كما أنه لا يستند إلى محاكي شبكي حيث قام الباحثون بنمذجة الوصلات الشبكية من خلال معاملين هما عرض حزمة الوصلة والتأخير (flow based) إن هذا الاسلوب من النمذجة جيد ولكن يمكن أن يكون غير دقيق في حال وجود تدفقات محدودة بسرعة وصلة معينة. كما أن هذا المحاكي لا يدعم أي آلية لتحمل الفشل.

RUPredHadoop : قام Shangming Ning وآخرون بتقديم RUPredHadoop [29] نموذج رياضي من أجل توقع انشغال الموارد في عناقيد المقابلة والاختزال. يقوم هذا النموذج بتوقع انشغال الموارد (الذاكرة، المعالج، الشبكة، القرص الصلب) من أجل مهمة واحدة ومن ثم وباستخدام بعض الحسابات الرياضية يتم استنتاج درجة انشغال كل مورد من أجل عدة مهام. وقد أثبت التجارب التي قام بها الباحثون أن هذه النموذج أعطى نسبة خطأ لا تتجاوز 10% عندما تم تجريبه على عنقود حاسوبي مؤلف من 80 عقدة ومن أجل ترتيب TB100 من البيانات. إن هذا النموذج وللأسف لا يعطي تصور كامل عن أداء تطبيقات المقابلة والاختزال كما أن الاختبارات التي تمت عليه كانت من أجل عمل واحد ولكن وفي حال إرسال عدة أعمال في أوقات مختلفة سيصبح موضوع هذه الدقة موضع تساؤل نظرا لتنافس الأعمال على الموارد. إلا أن هذا النموذج مفيد فهو يعطي تصور جيد لاستهلاك الموارد بالنسبة لكل عمل على حدا.

وفيما يلي الجدول (3–2) والذي يبين أهم الفروقات بين المحاكيات التي تم ذكرها خلال هذا الفصل.

الاسم	دعم الجدولة	عقد مفصة	المكتبات المستخدمة والتحقيق	محاكاة الشبكة	محاكاة عدة أعمال في نفس الوقت	تحمل الفشل	إصدار Hadoop	نوع المحاكاة
Mrperf [12]	FIFO	نعم	,tcl,c++, Python	NS2	نعم	نعم	1	DES
MRSim[18]	FIFO,fair	نعم	Simjava ,java	GridSim	نعم	لا	1	DES
MRtune[19]	لا يدعم	لا	غير معروف	غير معروف	لا	لا	1	غير معروف
HSim[21]	FIFO,fair	نعم	Java	لا يوجد	نعم	لا	1	DES
MRSG[26]	FIFO	لا	C	SimGrid	نعم	لا	1	DES
YARNSim [23]	FIFO	نعم ،لكن هنالك محدودية بحساب تأخير المعالجة	C ,ROSS	CODES	نعم	نعم	2	PDES
WaxElephant[28]	FIFO وFair وCapacity	نعم	Java	لا يوجد	نعم	لا	1	DES

الجدول (3–2) مقارنة بين أهم محاكيات المقابلة والاختزال

إن المقارنة السابقة دفعتنا لتصميم المحاكي HSG (الفصل الرابع) الذي يلحظ الأمور التالية:

- استخدام نموذج برمجة تفرعية يعتمد على تمرير الرسائل باستخدام نموذج العمال Actor model
- نموذج يقبل الاضافة والتوسعة البرمجية
- يجب أن يدعم نسخة Map–Reduce 2.
- قادر على محاكاة عقد مخصصة من حيث القرص الصلب والمعالج.
- قادر على محاكاة الشبكة الحاسوبية بشكل جيد والتي تعتبر عامل مهم في أداء المقابلة والاختزال.
- يجب أن يقدم المحاكي سرعة في تنفيذ عملية المحاكاة.
- يجب أن يقدم المحاكي آلية لتسهيل عملية إعداد وإجراء التجارب.
- يجب أن يقدم المحاكي آلية للحصول على النتائج والمتحولات الإحصائية بشكل سهل وواضح.
- يجب أن يكون قادرا على محاكاة عدة أعمال في نفس الوقت.
- يجب أن يدعم عدة خوارزميات جدولة.
- دعم تحمل الفشل

3-4 الخاتمة

تناول هذا الفصل أهم محاكيات المقابلة والاختزال حيث بينا ضمنه عدم قدرة محاكيات الحوسبة الشبكية على محاكاة أعمال المقابل والاختزال وبالتالي لا بد من حلول جديدة قد تعتمد على محاكيات الحوسبة الشبكية أو على محاكيات شبكية أخرى. و يعتبر المحاكي MRPerf أول محاك في هذا المجال تبعته أبحاث عدة و بتوجهات مختلفة فبعض المحاكيات قادرة على محاكاة تفاصيل كثيرة و تحقق الكثر من الميزات (التكرار، تحمل الفشل، أنماط الجدولة) مثل MRSim و MRPerf أما بعض الباحثين اكتفوا بإعطاء كلفة عامة على كل مهمة من مهام المقابلة أو الاختزال بحيث يؤمن توسعية كبيرة و سرعة أكبر كما هو الحال في MRSG، كما قدم بعض الباحثين جهوداً مهمة حيث ابتكروا نماذجا رياضية تعطي معلومات عن سلوك المقابلة و الاختزال مثل RUPredHadoop. ختمنا هذا الفصل بجدول يقارن بين أهم هذه المحاكيات من حيث أسلوب التحقيق والميزات التي يدعمها كل محاكي.

الفصل الرابع

تصميم المحاكي HSG وتحقيقه

4–1 المقدمة

إن السبب الرئيسي في تطوير محاك لإطار المقابلة والاختزال هو محدودية محاكيات المقابلة والاختزال من حيث دعم النسخة 2 من هذا الإطار ومن حيث أنماط الجدولة والتوسع. سنقدم في هذا الفصل التصميم المقترح لـ (Hadoop SimGrid) HSGوسنتكلم عن تحقيقه.

4–2 المتطلبات الواجب توافرها في المحاكي

بعد دراسة محاكيات المقابلة والاختزال في فصل الدراسة المرجعية والوقوف عند نقاط القوة فيها ونقاط الضعف خلصنا إلى المتطلبات التالية والواجب لحظها عند تحقيق المحاكي HSG.

1– قابلية التوسع: يجب أن يكون المحاك قابل للتوسعة بشكل كبير بحيث نستطيع اختبار أثر التوسعية على أداء المقابلة والاختزال.

2– يدعم نسخة Map–Reduce 2.

3– إمكانية محاكاة عقد مخصصة من حيث القرص الصلب والمعالج.

4– القدرة على محاكاة الشبكة الحاسوبية بشكل جيد والتي تعتبر عامل هام في أداء المقابلة والاختزال.

5– السرعة في تنفيذ عملية المحاكاة.

6– تسهيل عملية إعداد وإجراء التجارب.

7– الحصول على النتائج والمتحولات الإحصائية بشكل سهل وواضح.

8– محاكاة عدة أعمال في نفس الوقت.

9– دعم عدة خوارزميات جدولة.

4–3 مراحل تحقيق المتطلبات

1– اختيار المحاكي الشبكي الذي سيشكل البنية التحتية للمحاكي HSG.

2– وضع تصميم للمحاكي.

3– تحقيق المحاكي.

4–3–1 اختيار المحاكي الشبكي الذي سيشكل البنية التحتية للمحاكي HSG

إن Hadoop هو إطار مفتوح المصدر يعنى بالمعالجة الموزعة للبيانات الضخمة على العناقيد الحاسوبية التي من الممكن أن تتوسع لآلاف من العقد[30]. وعلى ذلك ولمحاكاة Hadoop فلابد من اختيار محاك شبكي يؤمن الدقة والتوسعية والسرعة في نفس الوقت.

إن الأبحاث على الشبكات اعتمدت على المحاكيات منذ زمن ليس بالقصير، وهذا أدى إلى ظهور العديد من المعايير والمحاكيات أمثال NS2، Omnet++, Opnet, SimGrid, GridSim

ويمكن تصنيف المحاكيات من حيث النماذج كالتالي[31, pp. 103–104]

- **النماذج على مستوى الحزمة (Packet–level Models)**

مثل NS2[32], OMNET++[33], OPNET[34]. إن المحاكيات الشبكية التي تستخدم نماذج على مستوى الحزمة هي محاكيات أحداث متقطعة تقوم بمحاكاة مجموعة البروتوكولات أي أن إرسال أو استقبال حزمة يعتبر حدثا. وبالتالي فإن دقة هذه المحاكيات تأتي من النمذجة الجيدة لكامل النظام المراد محاكاته. ولكن ولسوء الحظ فإن هذا النوع من المحاكاة بطيء جدا بالتالي هنالك محدودية في التوسع بدراسة نظم الحوسبة الموزعة واسعة النطاق. وعلاوة على ذلك فإن النماذج الجيدة جدة لا تفيد كثيرا في التطبيقات الكبيرة، حيث لا بد من تجهيز متحولات غير ضرورية. إن مثل هذه النماذج غير مستقرة وغير ملائمة للتطبيقات الكبيرة. بالتالي فإن التطبيقات الموزعة واسعة النطاق تعتمد وبشكل أساسي على نماذج أبسط.

- **النماذج التي تعتمد على التأخير (Delay–based).**

مثل LogGOPSim[35] و LogGPS[36] في هذه النماذج يتم تمثيل وقت الاتصال كتأخير ثابت أو يمكن أن يكفي استخدام توزيعات إحصائية. إن هذا النموذج قابل للتوسع بشكل كبير كما أن استهلاكه الذاكري من رتبة O(n) أي أنه يتعلق بشكل دقيق بعدد العقد، كما أن الزمن اللازم لحساب التأخير من رتبة O(1) فهو زمن ثابت. إن هذا النموذج مستخدم بشكل كبير في الحوسبة الموزعة واسعة النطاق ولكنه يفترض عدم حدوث ازدحام في الشبكة كما يفترض أن عرض الحزمة المستخدم كبير جدا.

- **النماذج على مستوى التدفق (flow level)**

حيث يتم اعتبار التدفق أو الاتصال ككيان واحد بدل من اعتباره عدد من الحزم المنفصلة. ويكون الوقت اللازم لنقل رسالة $T_{ij}(S)$ ذات الحجم S من المضيف i إلى المضيف j يعطى بالعلاقة

$$T_{ij}(S) = L_{ij}+S/B_{i,j} \quad (1-4)$$

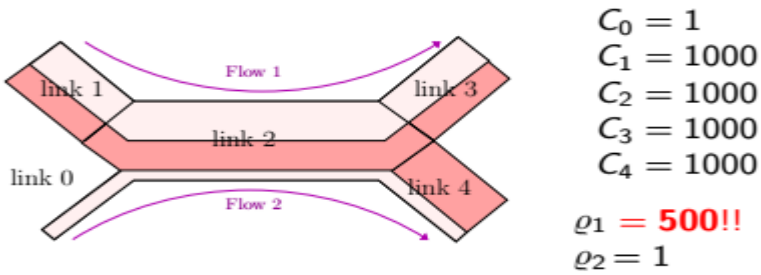
L_{ij}: هو التأخر الشبكي بين المضيف i والمضيف j.

B_{ij}: هو عرض الحزمة بين المضيفين.

إن توقع Lij يعتبر سهلا ولكن توقع B_{ij} أصعب لأنه يتطلب الأخذ بالحسبان التدفقات التي تجتاز نفس الوصلة. فعند بدء تدفق أو الانتهاء من تدفق يجب إعادة احتساب عرض الحزمة المخصص لكل تدفق، إن عملية إعادة الاحتساب غير مستهلة للذاكرة حيث أنها لا تعتمد على المعلومات السابقة الخاصة بعرض الحزمة المخصصة لكل تدفق. إن استخدام هذه الطريقة يقلل من الزمن اللازم للمحاكاة وتعتبر خيارا جيدا من أجل نمذجة التطبيقات الموزعة واسعة النطاق والتي تشتمل على كمية كبيرة من بيانات الاتصال.

لكن تبين أن هذا النموذج يؤثر على أداء بروتوكول TCP. وذلك لأن بروتوكول TCP يستخدم طريقة SLOW START فلا يتم استخدام كامل عرض الحزمة بشكل مباشر وإنما يتم التعديل على عرض الحزمة من أجل كشف ضياع الحزم. إن ذلك يكسر العلاقة الخطية المقدمة في المعادلة (1-4). ولكن يمكن معايرة الأداء بحيث نحصل على أداء جيد من أجل الرسائل كبيرة الحجم.

في النماذج على مستوى التدفق تتقاسم التدفقات عرض حزمة كل الوصلات بشكل عادل وهذا مخالف للواقع فمن الممكن أن يكون أحد التدفقات محدود بوصلة معينة بالتالي لا يشغل كامل الحصة المخصصة له كما في الشكل (1-4).



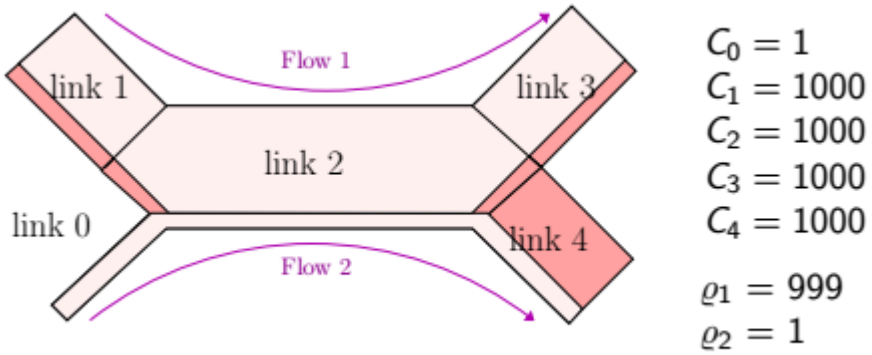
الشكل (4–1) مثال عن النموذج على مستوى التدفق[37]

فالتدفق الثاني محدود بالوصلة C0 وسيكون عرض الحزمة الخاص به 1، على الرغم من تخصيص نصف عرض حزمة الوصلة الثانية له ،بالتالي هنالك 499 غير مستخدمة كان يمكن أن يعطى هذا العرض من الحزمة للتدفق Flow1. تم حل هذه المحدودية باستخدام النموذج من نوع Fluid المستخدم في المحاكي SimGrid [15].

النموذج Fluid هو من النماذج على مستوى التدفق وهو يعتمد على نموذج المشاركة Max–Min.

إن نموذج Max–Min يهدف لزيادة عرض الحزمة الخاص بكل تدفق، وهنالك عدة خوارزميات من أجل ذلك أبسطها خوارزميةBucket–filling [37] ويمكن تطبيق هذه الخوارزمية بالشكل التالي:

- يتم إعطاء كل تدفق عرض حزمة مساوي للصفر.
- يتم زيادة كل عرض حزمة بمقدار صغير وبشكل متكرر.
- عندما يصبح عرض حزمة أحد التدفقات مساويا عرض حزمة وصلة يجتازها (تم اشباع الوصلة) يتم حذف التدفقات التي تجتاز هذ الوصلة من الحساب.
- يتم تكرار ذلك حتى تشبع كل الوصلات.



الشكل (4–2) مثال عن النموذج من نوع Fluid [37]

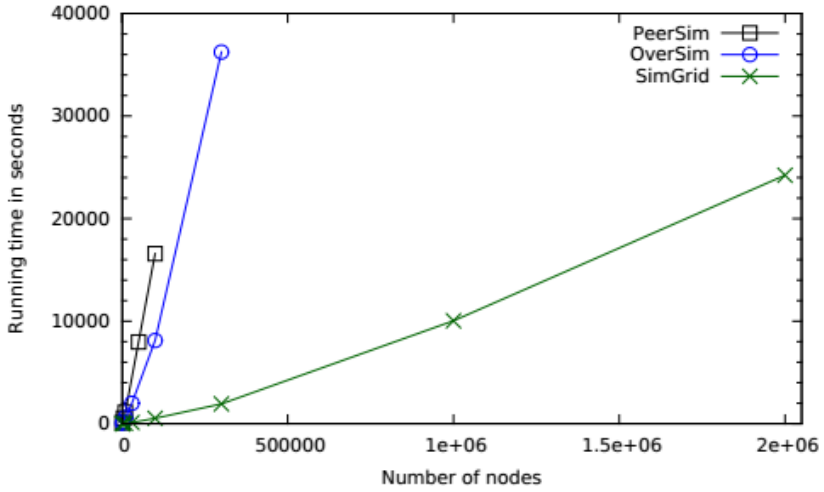
فالتدفق 2 محدود بالوصلة 1 بالتالي عرض الحزمة التي سيحصل عليها هو 1 أما التدفق 1 أصبح محدود الوصلة 2 وبالتالي سيحصل على باقي عرض حزمة الوصلة 2 وهو 999. وفي الجدول (4–1) مقارنة بين أهم المحاكيات الشبكية من حيث النموذج.

نوع النموذج	اسم المحاكي
على مستوى الحزمة	Ns–2
على مستوى الحزمة	Ns–3
على مستوى الحزمة	Omnet++
النموذج يعتمد على التأخير	OverSim
النموذج يعتمد على التأخير	GridSim
النموذج يعتمد على التأخير	CloudSim
Fluid	SimGrid

الجدول (4–1) مقارنة بين المحاكيات

قام Martin Quinson وآخرون[38] بمقارنة أداء SimGrid و

(Simulator based on Omnet++) OverSim وPeerSim وذلك بمحاكاة بروتوكول Chord[1] في شبكات الند للند، حيث قام الباحثون بدراسة تأثير زيادة عدد العقد على زمن المحاكاة وخلصوا إلى الشكل (4–3).



[1] بروتوكول وخوارزمية من أجل جداول التقطيع الموزعة (distributed hash tables) حيث يتم تخزين عدد من المفاتيح على كل عقدة.

الشكل (4–3) علاقة عدد العقد بسرعة المحاكاة عند محاكاة بروتوكول Chord وذلك بالنسبة لعدة محاكيات [20,p.127].

من الشكل(4–3) نرى أن SimGrid قادر على محاكاة عدد من العقد يصل إلى مليوني عقدة وبسرعة تصل إلى 15 ضعف مقارنة بباقي المحاكيات المستخدمة ضمن التجربة.

و من أجل التحقق من دقة SimGrid قام Henri Casanova وآخرون[39] بمقارنة أداء SimGrid بأداء المحاكي GTNetS من حيث زمن التأخير في الشبكة حيث اعتبروا أن نتائج المحاكي GTNetS صحيحة 100% على اعتبار أنه محاكي على مستوى الحزمة. وخلصت تجاربهم إلى النتائج التالية:

- إن دقة SimGrid تكون ضعيفة من أجل الرسائل القصيرة وذلك لأنه لا يأخذ بالحسبان آلية slow start في TCP .
- عندما يكون عرض حزمة التدفق محدود بعرض حزمة الوصلة كان مستوى الخطأ لا يتجاوز 5% وعندما يزداد الازدحام في الشبكة تقترب دقة SimGrid من المثالية.
- عندما يكون عرض حزمة التدفق محدود بزمن الذهاب والإياب تكون نسبة الخطأ لا تتجاوز 1%.

وباعتبار أن إطار المقابلة والاختزال الذي سنقوم بمحاكاته يهتم بمعالجة البيانات الضخمة ستكون دقة SimGrid مقبولة فيه نظرا لحجوم البيانات الكبيرة.

وعلى ذلك فإن SimGrid يعتبر مناسبا جدا لمحاكاة تطبيقات المقابلة والاختزال (توسعية كبيرة وسرعة كبيرة). بالتالي اعتمدنا على المحاكي SimGrid في بناء وتحقيق المحاكي HSG.

إن SimGrid هو إطار من أجل تطوير محاكيات للتطبيقات الموزعة التي يتم تنفيذها على بيئات موزعة، بحيث تستخدم هذه المحاكيات من أجل تقييم ومقارنة إعدادات البيئات وتصميم النظم والخوارزميات. تعتبر SimGrid متعددة الاستخدام لأنها تؤمن نماذج وواجهات برمجية APIs من أجل محاكات النظم الموزعة الشهيرة (الشبكات المحلية والشبكات الواسعة والعناقيد الحاسوبية ومراكز البيانات) وتتميز بالتالي:

1– الدقة: تم التحقق من دقة SimGrid بشكل نظري وعملي.

2– قابلية التوسع: إن نموذج المحاكاة في SimGrid والمحاكيات المبنية بالاعتماد عليها تعتبر سريعة وغير مستهلكة للذاكرة بشكل كبير بحيث يمكن تشغيلها بشكل سريع على عقدة واحدة.

3– الاستخدام: إن SimGrid برمجية مفتوحة المصدر و مجانية و يمكن تشغيلها على نظم التشغيل (Windows , Max OS and Linux) و يمكن للمستخدم كتابة المحاكيات الخاصة به باستخدام عدة لغات برمجة (C++,Python or Java)

إن أي برنامج محاكاة يعتمد على SimGrid يتألف من عدد من العمال (Actors) والتي تقوم بتنفيذ دوال المستخدم. تقوم هذه الدوال باستخدام الواجهة s4u من SimGrid من أجل تنفيذ عمليات الاتصال والحساب واستخدام القرص الصلب. هذه النشاطات يتم تنفيذها على الأجهزة والوصلات والأقراص.

وفيما يلي مثال بسيط من نوع ping pong سنبين من خلاله مكونات برنامج SimGrid:

1– ملف هيكلية الشبكة: وهو عبارة عن ملف xml نحدد فيه الوصلات والمعالجات والأجهزة وقد تحوي على أقراص صلبة وفيما يلي النص البرمجي للملف network.xml

```xml
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <zone id="AS0" routing="Full">
    <host id="bob" speed="1Gf">
      <prop id="ram" value="100B" />
      <disk id="Disk1" read_bw="100MBps" write_bw="40MBps">
        <prop id="size" value="500GiB"/>
        <prop id="mount" value="/scratch"/>

      </disk>
      <disk id="Disk2" read_bw="200MBps" write_bw="80MBps"/>
    </host>

    <host id="alice" speed="1Gf">
      <disk id="Disk1" read_bw="200MBps" write_bw="80MBps">
        <prop id="content" value="storage/content/small_content.txt"/>
      </disk>
      <prop id="ram" value="100B" />
    </host>

    <host id="carl" speed="1Gf">
      <prop id="remote_disk" value="/scratch:Disk1:bob"/>
    </host>

    <link id="link1" bandwidth="125MBps" latency="150us" />
    <link id="link2" bandwidth="125MBps" latency="150us" />
    <link id="link3" bandwidth="125MBps" latency="150us" />

    <route src="bob" dst="alice">
      <link_ctn id="link1" />
    </route>
    <route src="bob" dst="carl">
      <link_ctn id="link2" />
    </route>
    <route src="alice" dst="carl">
      <link_ctn id="link3" />
    </route>
  </zone>
</platform>
```

2– النص البرمجي ويحتوي على ملف أو عدة ملفات مكتوبة بلغة c++ أو Python أو Java و فيما يلي الملف PingPong.cpp و هو يحتوي على Actor اسمه pinger يقوم بإرسال رسالة إلى Actorآخر اسمه ponger يقوم باستقبال الرسالة و إعادة إرسالها. عمليا يمكن ربط كل Actor بصندوق بريد وهو عبارة عن غرض تقوم ال Actors بإرسال الرسائل إليه كم يمكن لل Actors أن تنتظر وصول رسالة من نوع معين إلى صندوق البريد.

كل برنامج يحتوي على دالة main تقوم بإنشاء نسخة من محرك المحاكاة الخاص ب SimGrid وتقوم بتحميله بملفات الشبكة والنشر.

```
#include <simgrid/s4u.hpp>
namespace sg4 = simgrid::s4u;

XBT_LOG_NEW_DEFAULT_CATEGORY(s4u_app_pingpong, "Messages specific for this s4u example");

static void pinger(sg4::Mailbox* mailbox_in, sg4::Mailbox* mailbox_out)
{
  XBT_INFO("Ping from mailbox %s to mailbox %s",
  mailbox_in->get_name().c_str(), mailbox_out->get_name().c_str());

  /* - Do the ping with a 1-Byte payload (latency bound) ... */
  auto* payload = new double(sg4::Engine::get_clock());

  mailbox_out->put(payload, 1);
  /* - ... then wait for the (large) pong */
  auto sender_time = mailbox_in->get_unique<double>();

  double communication_time = sg4::Engine::get_clock() - *sender_time;
  XBT_INFO("Payload received : large communication (bandwidth bound)");
  XBT_INFO("Pong time (bandwidth bound): %.3f", communication_time);
}
```

```
static void ponger(sg4::Mailbox* mailbox_in, sg4::Mailbox* mailbox_out)
{
  XBT_INFO("Pong from mailbox %s to mailbox %s",
  mailbox_in->get_name().c_str(), mailbox_out->get_name().c_str());

  /* - Receive the (small) ping first ....*/
  auto sender_time          = mailbox_in->get_unique<double>();
  double communication_time = sg4::Engine::get_clock() - *sender_time;
  XBT_INFO("Payload received : small communication (latency bound)");
  XBT_INFO("Ping time (latency bound) %f", communication_time);

  /*  - ... Then send a 1GB pong back (bandwidth bound) */
  auto* payload = new double(sg4::Engine::get_clock());
  XBT_INFO("payload = %.3f", *payload);

  mailbox_out->put(payload, 1e9);
}
int main(int argc, char* argv[])
{
  sg4::Engine e(&argc, argv);
  e.load_platform(argv[1]);
  e.load_deployment(argv[2]);
  sg4::Mailbox* mb1 = sg4::Mailbox::by_name("Mailbox 1");
  sg4::Mailbox* mb2 = sg4::Mailbox::by_name("Mailbox 2");

  e.run();

  XBT_INFO("Total simulation time: %.3f", sg4::Engine::get_clock());

  return 0;
}
```

3– يتم توزيع ال Actor على الشبكة من خلاص ملف من نوع xml بحيث نحدد الActors التي ستعمل على عقدة معينة، وفيما يلي الملف platform.xml

```
<platform version="4.1">
  <actor host="alice" function="pinger"  />
  <actor host="bob" function="ponger" />
</platform>
```

إن أسهل طريقة لتشغيل برنامج SimGrid هو تنزيل نموذج من الرابط https://framagit.org/simgrid/simgrid-template-s4u وإضافة الملفات السابقة إلى هذا النموذج والتعديل على الملف CmakeList.txt بحيث نضيف عليه الملفات ذات اللاحقة cpp لتتم ترجمتها.

```
cmake_minimum_required(VERSION 2.8.8)

project(hsg) # TODO: give a real name to your project here

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 ")
set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} "${CMAKE_SOURCE_DIR}/cmake/Modules/")

find_package( SimGrid 3.21 REQUIRED) # This template requires SimGrid v3.25
# Locate GTest
find_package(GTest REQUIRED)


include_directories(${SimGrid_INCLUDE_DIR}
${GTEST_INCLUDE_DIRS}

)


add_executable(PingPong

Main.cpp
)
target_link_libraries(hsg ${SimGrid_LIBRARY}
 ${GTEST_LIBRARIES} pthread
)
```

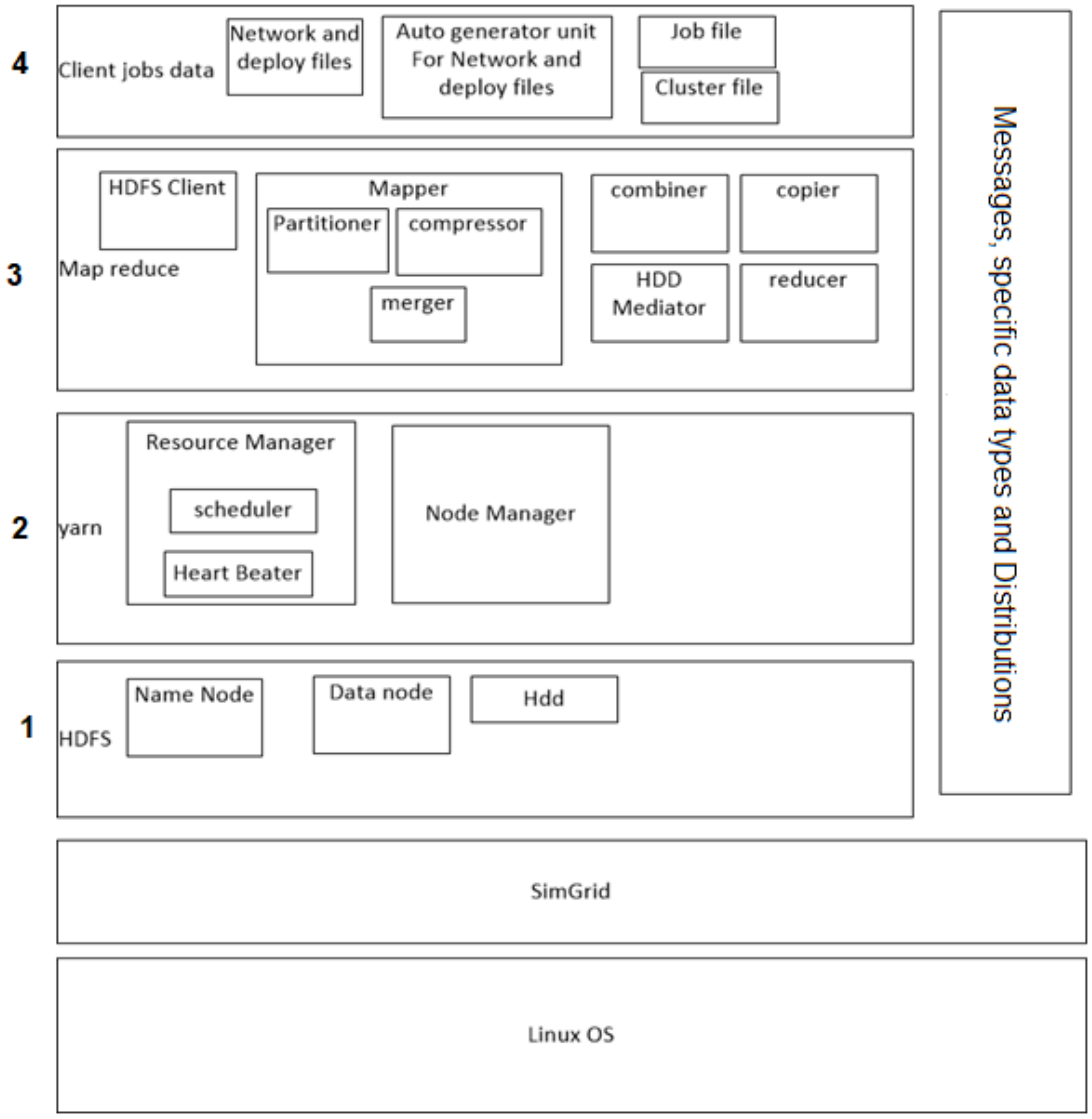
بعدها نقوم بتشغيل البرنامج بالشكل التالي بحيث نزود البرنامج بملفات الشبكة والنشر.

```
Cmake .
make
./PingPong network.xml     platform.xml
```

من اجل بناء المحاكي HSG قمنا باستخدام الاصدار 3.26 من SimGrid واستخدمنا لغة C++.

4–3–2 تصميم المحاكي HSG

عند وضع تصميم للمحاكي استفدنا من نموذج العمال (Actor Model) المستخدم في SimGrid وذلك في الفصل بين الوحدات الأساسية للمحاكي فهذه الوحدات تتواصل فيما بينها باستخدام الرسائل فقط ولا يوجد استدعاء للطرائق، وفي الشكل (4–4) البنية العامة للمحاكي HSG.



الشكل (4–4) البنية العامة للمحاكي HSG

نلاحظ من الشكل (4–4) أن المحاكي HSG تم تحقيقه فوق المحاكي الشبكي SimGrid الذي بدوره يعمل فوق نظام التشغيل Linux. إن HSG يتكون من الأقسام التالية:

1- HDFS (نظام ملفات Hadoop الموزع): وهو مسؤول عن كتابة وقراءة الملفات والتحقق من تكاملها وتكراراها ويتكون من:

a. عقدة الأسماء (Name Node) وهي وحيدة بالنسبة للعنقود ومسؤولة عن تكرار البيانات وتحتوي معلومات تبين أماكن توزع أقسام الملفات.

b. عقد البيانات (Data Nodes) يتم تشغيلها على الأجهزة المضيفة وتقوم بتخزين وقراءة البيانات إلى الأقراص الصلبة المحلية.

c. القرص الصلب Hdd وهو عقدة تمثل القرص الصلب.

2- YARN(yet another resource negotiator) والذي بدوره يحتوي على نوعين من العقد:

a. مدير العقدة وهي عقدة وحيدة من أجل كل مضيف تدير الحاويات (Containers) التي تقوم بالتنفيذ على هذا المضيف علما أن هذه الحاويات يمكن أن تكون إما مقابل أو مختزل أو تطبيق رئيسي.

b. .مدير الموارد وهو عقدة وحيدة من أجل كامل العنقود مسؤول عن جدولة الأعمال ومتابعة تنفيذها كما هو مسؤول عن حجز الموارد (الحاويات Containers).

3- تطبيق المقابلة والاختزال (Map–Reduce) ويتكون من المكونات التالية.

a. المقابل (Mapper) وهو عقدة يتم إنشاؤها من قبل مدير العقدة وعادة يتم إنشاء مقابل واحد من أجل كل جزء من البيانات (Chunk) يحتوي المقابل بدوره على عدد من المكونات وهي المقسم 1–(partitioned) الذي يقوم بتقسيم خرج المقابل على كل المختزلات (Reducers) 2–الدامج (merger) الذي يقوم بدمج خرج المقابل 3–الضاغط الذي يقوم بضغط الخرج من أجل توفير المساحة عند التخزين والنقل عبر الشبكة

b. المختزل (Reducer) وهو عقدة يتم إنشاؤها من قبل مدير العقدة وتستقبل خرج المقابل (Mapper output) وتجري عليه بعض العمليات لتحصل على نتيجة نهائية تقوم بكتابتها إلى نظام الملفات الموزع كما يحتوي المختزل على دامج وفاك للضغط (decompress).

c. المجمع (Combiner) يقوم وعند استخدامه باختصار حجم البيانات وعادة عمله يشبه عمل المختزل ولكن يطبق على جزء من البيانات وذلك عكس المختزل الذي يطبق على كامل البيانات.
d. الناسخ (Copier) وهو مسؤول عن نسخ خرج المقابل إلى المختزل عبر الشبكة.
e. وسيط القرص الصلب (Hdd Mediator) وهو عبارة عن صف يساعد علميات المقابلة والاختزال في الوصول إلى القرص الصلب التي تتوضع عليه البيانات كما يساعدهم في الكتابة على القرص الصلب.

4– بيانات أعمال المستخدم: وتتألف من:

a. ملف البيئة الذي يحدد العنقود الحاسوبي ومواصفات الأجهزة فيه ونوع الوصلات.
b. ملف النشر الذي يحدد توزع العقد على الشبكة.
c. ملفات الأعمال وهي ملفات JSON توصف الأعمال المرسلة للمحاكي.
d. ملف العنقود وهو ملف JSON يقوم بتوصيف العنقود المستخدم وهذا الملف هو دخل لوحدة توليد الشبكة وملفات النشر (Auto generate unit for network and deploy files)
e. وحدة توليد الشبكة وملفات النشل (Auto generate unit for network and deploy files) تقوم بقراءة ملف العنقود وتولد ملفات البيئة والنشر مما يسهل التجارب على العنقود كما يمكن تزويد المحاكي بملفات (بيئة ونشر) بشكل منفصل عن هذه الوحدة وذلك من أجل محاكاة عناقيد حاسوبية تحتوي على مكونات غير متجانسة.

5– مكتبة تحتوي على عدد من التوزيعات العشوائية بالإضافة إلى عدد من أنماط المعطيات الخاصة بالمحاكي.

4–3–3 تحقيق المحاكي

يعتمد التحقيق الناجح لمثل هذا المحاكي على تصميم وتنفيذ دقيق للنماذج التي تمثل الموارد الرئيسية في كل عقدة، وهي بالتحديد القرص الصلب والمعالج والذاكرة. سنعرض هنا هذه التصاميم والتحقيقات إضافة إلى الانظمة المعتمدة عليها.

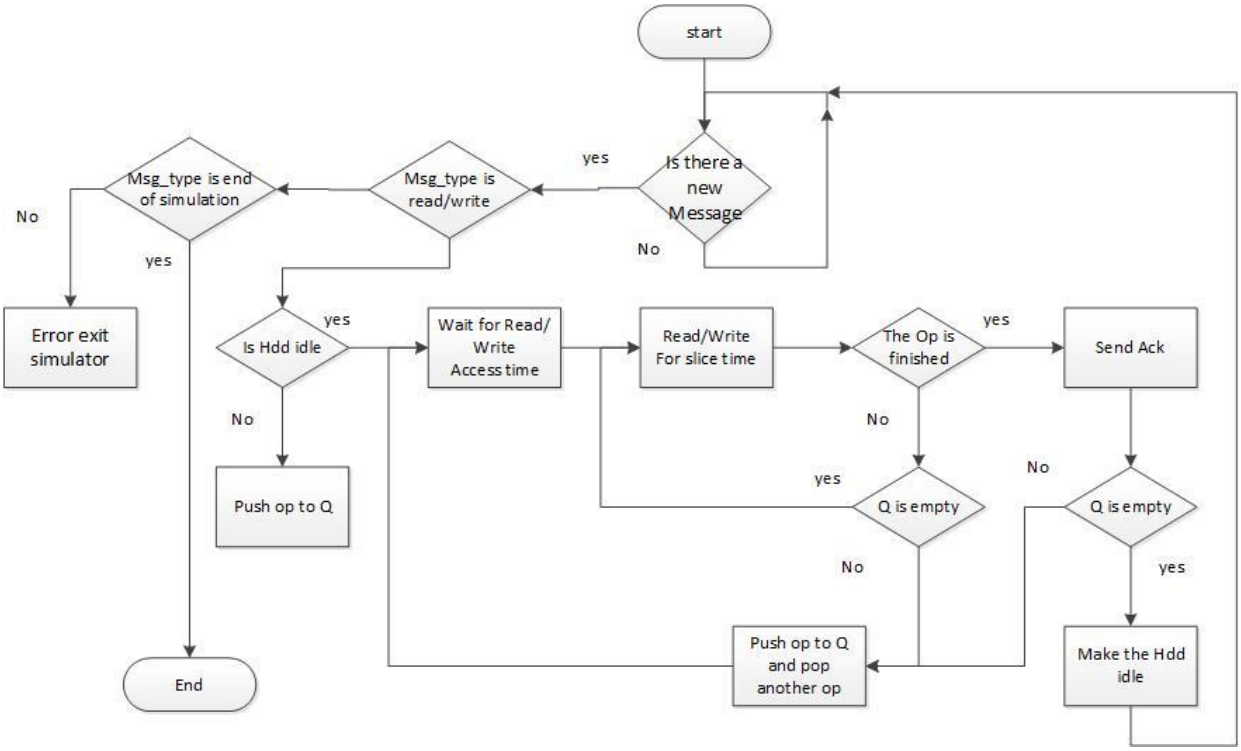
4–3–3–1 تحقيق الموارد الأساسية

المعالج إن المحاكي SimGrid يحتوي على نموذج للمعالج بحيث يمكن تحديد سرعة هذا المعالج وعدد النوى وذلك من أجل كل عقدة وقد اعتمدنا على هذا النموذج ضمن المحاكي HSG.

الذاكرة: قمنا باستخدام نموذج بسيط نسبيا يشتمل على القراءة و الكتابة على الذاكرة و يعتمد النموذج على سرعة القراءة و الكتابة و الحجم المراد قراءته أو كتابته و هذا النموذج هو نفس النموذج الذي تم استخدامه في المحاكي YARNSim [23] و باعتبار تأخير القراءة أو الكتابة على الذاكرة صفري نظرا لصغر هذه القيمة أمام باقي العمليات التي تستهلك زمن أكبر.

القرص الصلب: إن النموذج الموجود في SimGrid لا يناسب تطبيقات المقابلة والاختزال وذلك لأنه لا يأخذ بعين الاعتبار زمن الوصول للبيانات[40]

وإذا اعتبرنا أن الملفات مكتوبة على كتل متجاورة في القرص فإن قراءة الملف ستحتاج إلى زمن نقل البيانات من القرص مضافة إليه زمن الوصول لمرة واحدة. لكن وفي حال وجود أكثر من عملية قراءة وكتابة في نفس الوقت فإن رأس القراءة والكتابة سيتحرك بشكل متكرر لينفذ هذه العمليات بنفس الوقت بحيث ينفذ جزء من عملية قراءة أو كتابة لينتقل لينفذ جزء من عملية أخرى، إن ذلك يؤدي إلى زيادة زمن العملية. علما أن رأس القراءة والكتابة يعطي كل عملية 100 ميلي ثانية بشكل افتراضي قبل أن ينتقل لتنفيذ عملية أخرى. إن هذا النموذج أقرب إلى خوارزمية CFQ(Complete Fairness queueing) المستخدمة في نظام التشغيل Linux [41]. وفي الشكل (4–5) مخطط تدفقي يوضح آلية تحقيق عملية القراءة والكتابة على القرص الصلب.



الشكل (4–5) المخطط التدفقي لتسلسل تنفيذ عمليات القراءة والكتابة على القرص الصلب

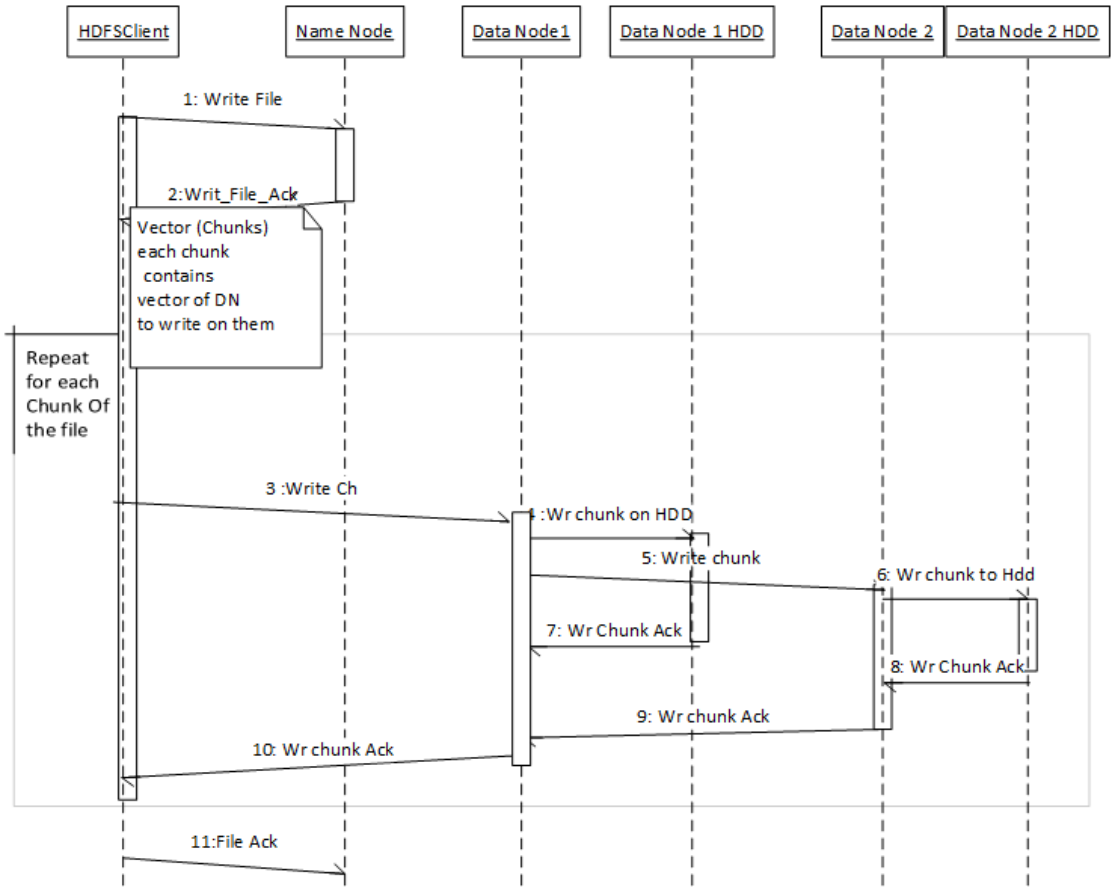
فعند إنشاء عقدة القرص الصلب تبدأ هذه العقدة بانتظار الرسائل وعند وصول رسالة معينة تختبر نوع الرسالة فإذا كانت من نوع end_simulation ينهي القرص الصلب عمله، أما إن كانت من نوع read/write فيتم تنفيذ جزء منها بشكل مباشرة على القرص الصلب في حال كان غير مشغول وفي حال العكس تضاف العملية إلى الرتل. بعد تنفيذ جزء من العملية يتم اختبار الرتل للمتابعة في التنفيذ أو الانتقال إلى عملية أخرى في حال وجود عمليات في الرتل. علما أنه في أي لحظة ممكن أن تصل عمليات جديدة للقرص الصلب ويكون تسلسل التنفيذ حسب وقت المحاكاة فعلى سبيل المثال إن كان هنالك عملية تنفذ على القرص الصلب وفي نفس الوقت وصلت رسالة جديدة يتم إضافة الرسالة الجديدة إلى الرتل بشكل مباشر (بما أن المحاكي يعمل بطريقة الأحداث المتقطعة).

4–3–3–2 نظام الملفات الموزع (HDFS)

قمنا بتحقيق نظام الملفات الموزع (HDFS) بحيث يدعم عمليات القراءة **4–3–3–2–1** والكتابة **4–3–3–2–1** والحذف. ولم نعنى بعمليات التعديل لأنه لا يتم استخدامها في تطبيقات المقابلة والاختزال، كما أن نظام الملفات الموزع يدعم تكرار البيانات وتوزيع النسخ على أكثر من رف (rack). وكل عقدة بيانات يمكنها أن تكتب وتقرأ على أكثر من قرص صلب.

4-3-3-2-1 عملية الكتابة على HDFS

وفي الشكل (4–6) مخطط UML تسلسلي يوضح آلية عملية الكتابة من أجل معامل تكرار 2.



الشكل (4–6) مخطط UML تسلسلي يبين آلية الكتابة على HDFS من أجل معامل تكرار 2

فعند كتابة ملف على نظام الملفات الموزع (HDFS) يتم استدعاء الدالة writeFile من الصف HDFSClient وخلف هذا الاستدعاء هنالك عدد من العمليات وفق الشكل (4–6):

1– يرسل HDFSClient رسالة من نوع HDFS_wr_file إلى عقدة الأسماء تحتوي هذه الرسالة معلومات عن الملف (الحجم والاسم والمجلد).

2– تقوم عقدة الأسماء بتحديد عدد أجزاء (Chunk)الملف من خلال تقسيم حجم الملف على حجم الجزء (بشكل افتراضي 128MB) وتقوم بتحديد عقد البيانات التي ستقوم بكتابة كل جزء وذلك تبعا لمعامل التكرار مع مراعات توزع الأجزاء على أكثر من عقدة وأكثر من رف. ثم تقوم عقدة الأسماء بإرسال شعاع من الأجزاء وكل جزء يحتوي على شعاع من عقد البيانات والتي ستتم عليها عملية الكتابة.

3- يقوم HDFSClientومن أجل كل جزء بإرسال رسالة من نوع wr_chunk إلى أول عقدة من عقد البيانات المسؤولة عن كتابة هذا الجزء.

4- تقوم عقدة البيانات بعد استلام الرسالة بتحديد القرص الصلب الذي ستقوم بالكتابة عليه ومن ثم ترسل للقرص الصلب طلب الكتابة.

5- وبشكل متزامن تقوم عقدة البيانات بإرسال طلب الكتابة إلى العقدة التالية المسؤولة عن كتابة نفس الجزء.

6- تقوم عقدة البيانات الثانية بإرسال طلب الكتابة إلى أحد الأقراص الصلبة المرتبط معها.

7- عند انتهاء عملية الكتابة على القرص الصلب يرسل القرص الصلب إشعار انتهاء إلى عقدة البيانات المرتبط معها.

8- بشكل مشابه للخطوة 7.

9- تقوم عقدة البيانات الثانية بإرسال إشعار انتهاء إلى العقدة التي أرسلة إليها طلب الكتابة.

10- عند وصول إشعار الانتهاء من عملية الكتابة إلى عقدة البيانات الأولى من العقدة الثانية ووصول إشعار الانتهاء من القرص الصلب تقوم العقدة الأولى بإرسال اشعار انتهاء من كتابة الجزء إلى HDFSClient.

11- الخطوات من 3 إلى عشرة يتم تكرارها من أجل كل جزء من الملف.

عند الانتهاء من كتابة كافة الأجزاء يرسل الصف HDFSClient إشعار انتهاء من عملية كتابة الملف إلى عقدة الأسماء.

4-3-3-2-2 عملية القراءة من نظام الملفات الموزع HDFS

في الشكل (4-7) مخطط UML تسلسلي يوضح آلية القراءة من HDFS فعند استدعاء الدالة ReadFile على الصف HDFSClient هنالك عدد من العمليات التي تتم وفق الترتيب التالي:

1- يقوم الصف HDFSClient بإرسال رسالة من النوع re_file إلى عقدة الأسماء.

2- تقوم عقدة الأسماء وفي حال وجود الملف بإرسال إشعار يحتوي على شعاع من الأجزاء التي تشكل هذا الملف وكل جزء يحتوي على شعاع فيه عقد البيانات التي تحتوي هذا الجزء.

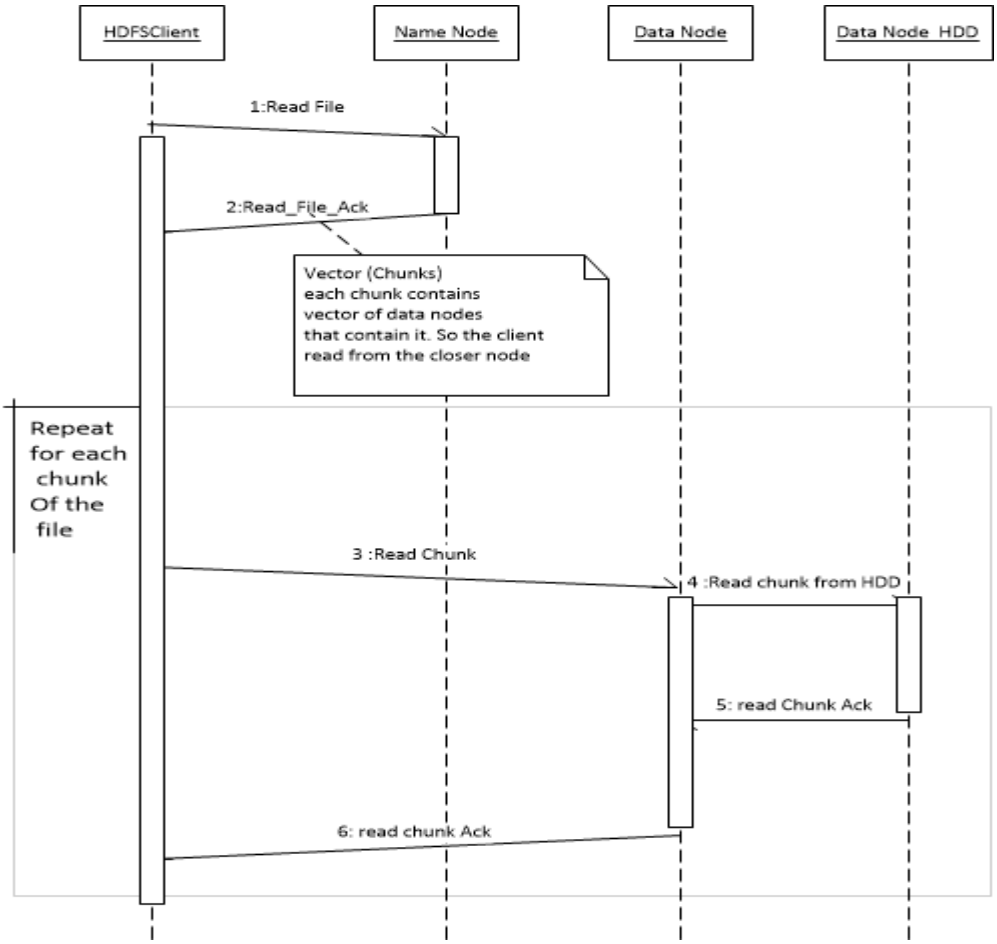
3– من أجل كل جزء يتم إرسال طلب قراءة إلى إحدى عقد الأسماء ويفضل أن تكون هذه العقدة على نفس المضيف وفي حال عدم وجود الجزء على نفس المضيف يفضل أن تكون ضمن نفس الرف وفي حال تعذر ذلك يتم اختيار عقدة بيانات بشكل عشوائي.

4– تقوم عقدة الأسماء بإرسال طلب قراءة إلى القرص الصلب الذي يحتوي على هذا الجزء.

5– عند انتهاء القرص الصلب من عملية القراءة يقوم بإرسال إشعار إلى عقدة الأسماء يبين فيه انتهاء عملية القراءة.

6– عندما يصل إشعار الانتهاء من القرص الصلب إلى عقدة البيانات تقوم بإرسال إشعار انتهاء من عملية القراءة إلى HDFSClient.

7– يتم تكرار الخطوات من 3 إلى 6 حتى الانتهاء من قراءة كامل أجزاء الملف.



الشكل (4–7) مخطط UML تسلسلي يمثل عملية القراءة من نظام الملفات الموزع.

4-3-3-2-3 حذف ملف من HDFS

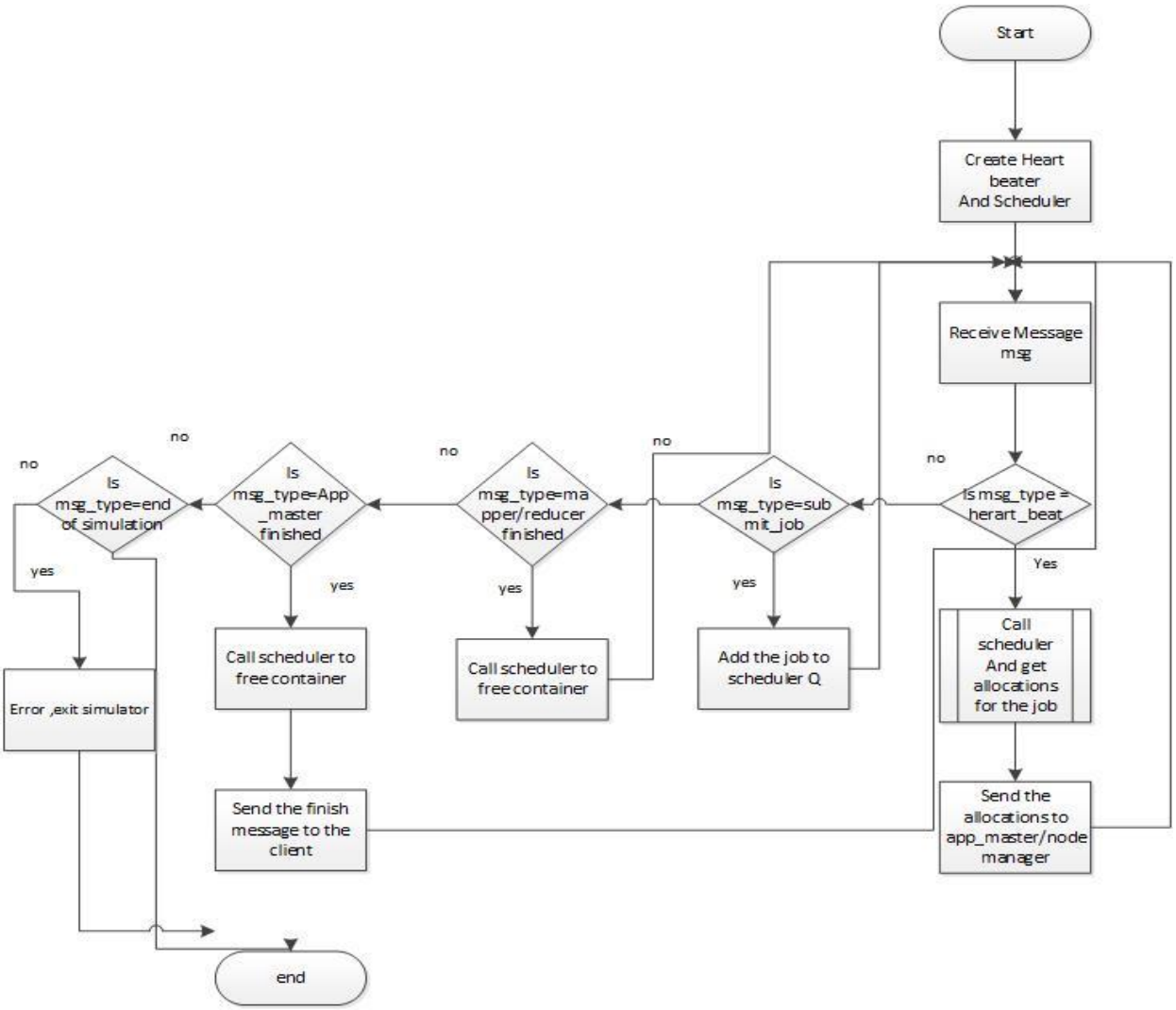
إن عملية الحذف بسيطة تتم حسب التسلسل التالي:

1- يرسل الزبون رسالة إلى عقدة الأسماء من أجل حذف ملف يحدد فيه اسم الملف ومساره.
2- تقوم عقدة الأسماء بإرسال رسالة الحذف إلى عقد البيانات التي تحتوي أجزاء هذا الملف.
3- تقوم عقد البيانات بحذف الأجزاء وإعلام عقدة الأسماء بذلك.
4- تقوم عقدة الأسماء بحذف السجل الخاص بهذا الملف وإعلام الزبون بنجاح عملية الحذف.

4-3-3-3 تحقيق YARN

عند تحقيق YARN قمنا بتحقيق نوعين من العقد:

1- مدير العقدة(Node manager) ودوره ينحصر في إنشاء وإنهاء الحاويات (Mapper,Reducer,App master) علما أن عدد الحاويات الكلي على كل مدير عقدة يساوي عدد نوى المعالجة مقسوما على العدد الذي يمثل numCorePerContainer و الذي نقوم بتحديده في الملف cluster.json. فعدما يستلم مدير العقدة رسالة من مدير المواد بخصوص انشاء حاوية يقوم بتشغيل Actor على هذه العقدة وإعطائه المعلومات الخاصة بالعمل ومكان تخزين بيانات المقابلة (في حال كانت العملية مقابلة). كما يقوم مدير العقدة بحذف حاوية حال انتهاء تنفيذها.
2- مدير الموارد (Resource Manager)مهمته جدولة الأعمال وإسناد الموارد والشكل (4-8) يبين لنا آلية عمل مدير الموارد:



الشكل (4–8) آلية عمل مدير الموارد.

حيث يقوم مدير الموارد وعند تشغيله بإنشاء Actor من نوع HeartBeater ومهمة هذا الأخير إرسال رسائل على شكل نبضات القلب إلى مدير الموارد (يعمل كمؤقت) تفصل بين هذه الرسائل فترات زمنية متساوية وبشكل افتراضيا 3 ثواني. بعدها وعندما يستلم مدير الموارد رسالة جديدة يقوم بفحص نوعها ونميز هنا الحالات التالية:

1– الرسالة من نوع heart_beat يتم استدعاء المجدول ومن ثم إرسال ردود حجز الموارد إلى التطبيق الرئيسي لعمل المقابلة أو الاختزال أو إلى مدير عقدة ليتم إنشاء تطبيق رئيسي.

2– الرسالة من نوع إرسال عمل(عمل مقابلة واختزال جديد) يتم إضافة العمل إلى رتل المجدول.

3– الرسالة تبين انتهاء عملية مقابلة أو عملية اختزل بالتالي يتم استدعاء المجدول لتحرير الحاوية المسؤولة عن هذه المهمة.

4– الرسالة تبين انتهاء تطبيق رئيسي بالتالي يتم استدعاء المجدول لتحرير الحاوية المسؤولة عن التطبيق ومن ثم يتم إرسال رسالة تبين انتهاء العمل إلى الزبون.

5– الرسالة من نوع end of simulation يتم انهاء مدير الموارد.

أما عن الآلية المتبعة لتحقيق المجدول (راجع الفصل الثاني الصفحة 54) فقد قمنا بتحقيق جميع أنواع الجدولة التي يدعمها Hadoopهي:

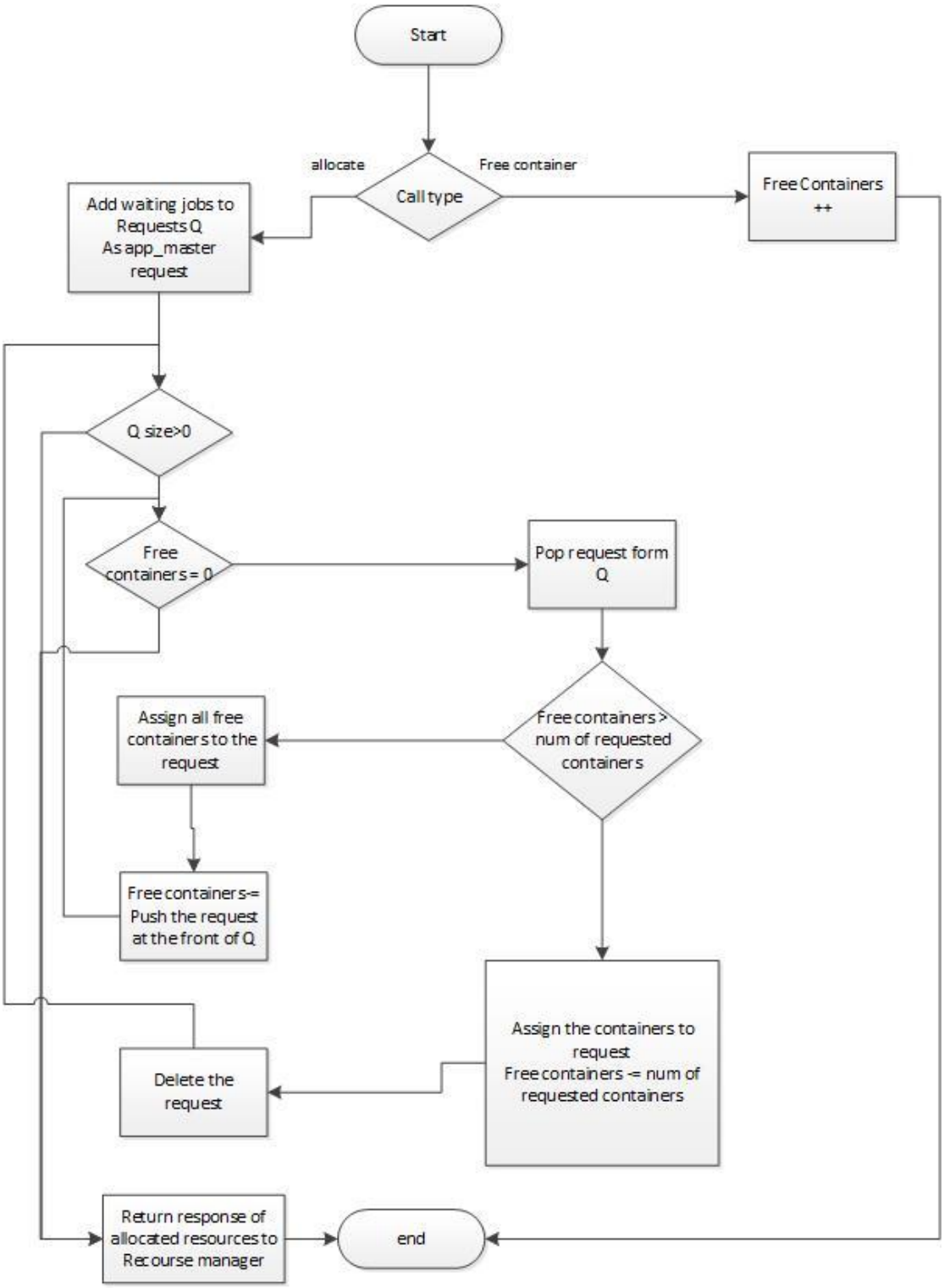
1– FIFO الداخل أولا خارج أولا أي يتم تنفيذ الأعمال بحسب ترتيب ورودها وعند الانتهاء من عمل ما يتم الانتقال إلى العمل التالي.

2– Fair أي المجدول العادل حيث يتم اسناد الموارد إلى الأعمال بشكل متساو.

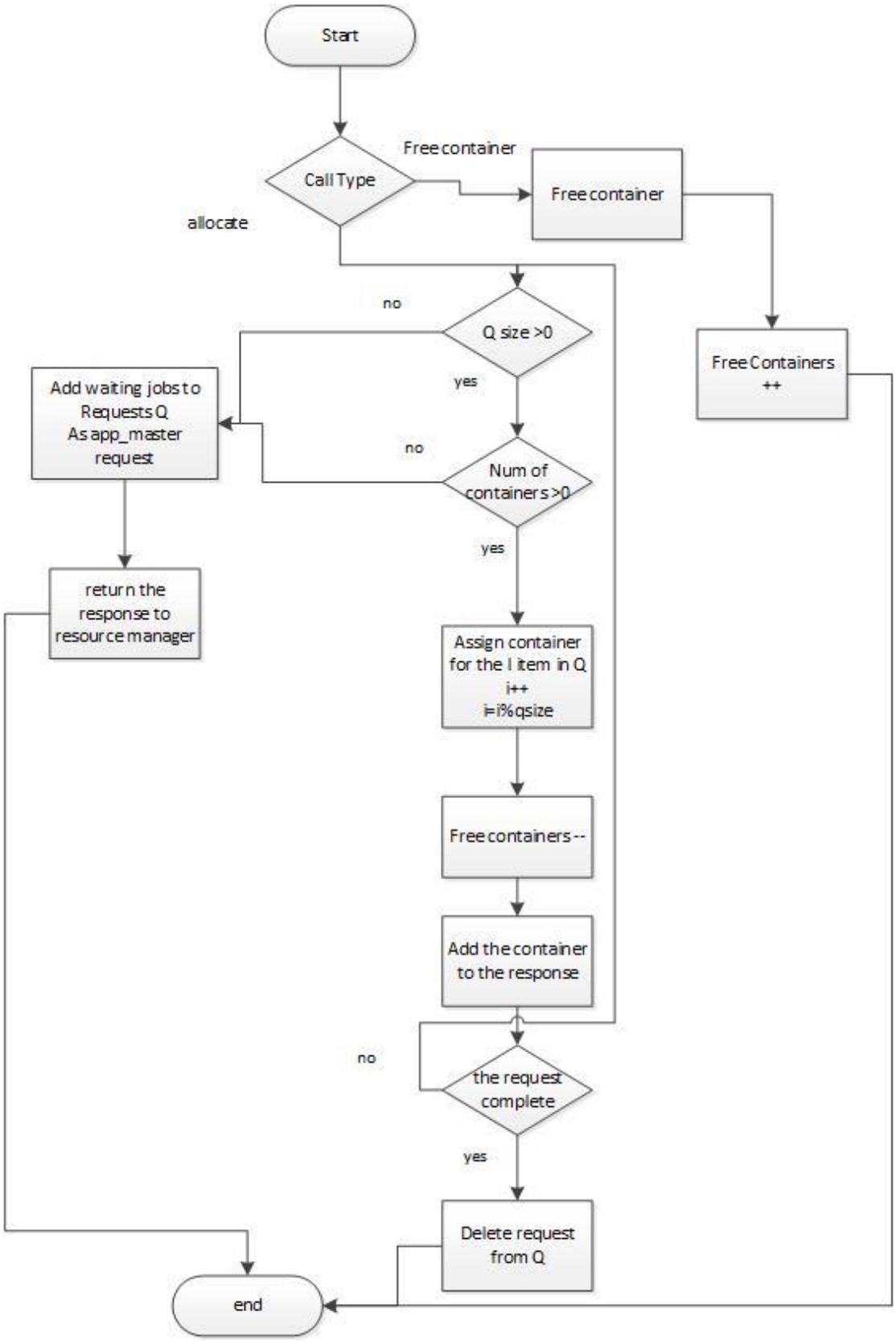
3– Capacity المجدول السعوي حيث يتم توزيع موارد العنقود على اكثر من رتل FIFO

و هذه الأنماط جميعها ترث الصف YarnSchedulerBase وتحقق الدوال jobAdd,finishJob,allocate,freeContainer

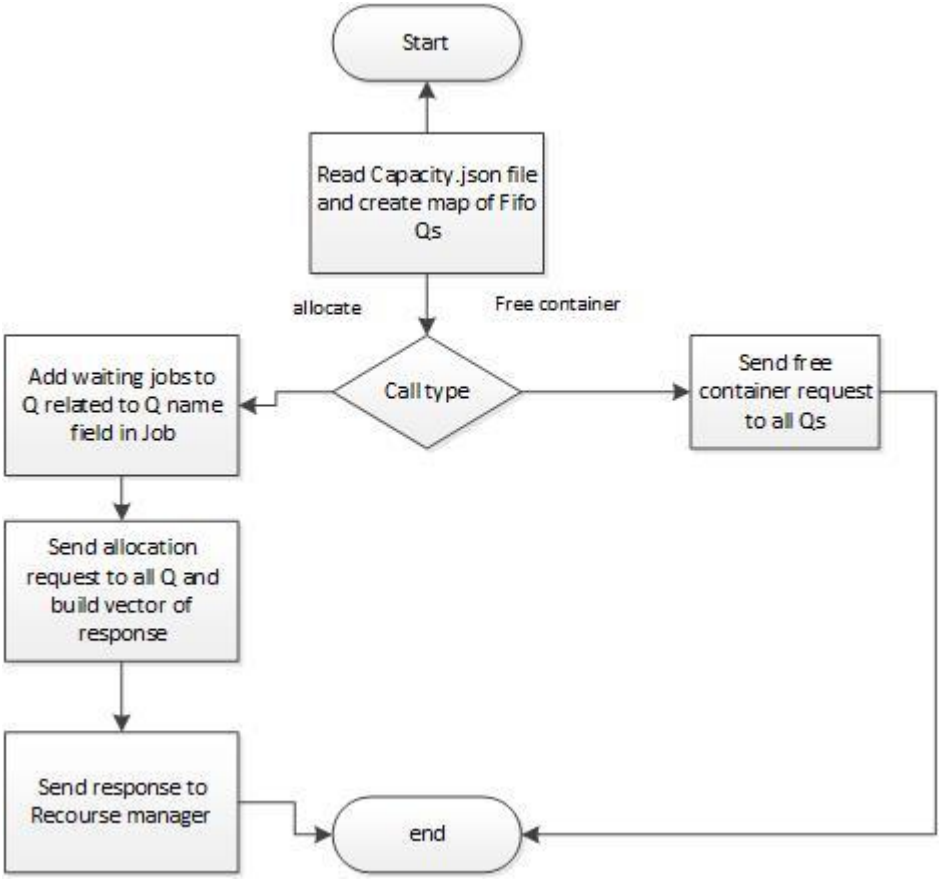
وفي الأشكال (4–9) و (4–10) و (4–11) آلية عمل المجدولات FIFO و FAIR و Capacity على الترتيب.



الشكل (4–9) المجدول FIFO



الشكل (4–10) المجدول Fair



الشكل (4–11) المجدول Capacity

إن الأشكال (4–9) و (4–10) و(4–11) تبين لنا أنه وفي حال استدعاء المجدول من أجل تحرير حاوية معينة فإنه يتم تحرير الحاوية ومن ثم يتم زيادة عدد الحاويات الحرة بمقدار واحد. أما في حال كان الاستدعاء من أجل الجدولة يتم اختبار نوع الجدولة وهنا نميز ثلاث حالات:

1– المجدول من نوع Fair الشكل (4–10): يتم اختبار الرتل بحيث يحتوي على طلبات حجز للموارد ومن ثم يتم اختبار وجود حاويات حرة وفي حال كان ذلك يتم إسناد حاوية لأول طلب ويتم تكرار العملية من أجل باقي الطلبات بشرط توفر حاويات حرة ووجود طلبات في الرتل علما أنه يتم حذف كل طلب بعد أن يتم إسناد كامل الحاويات التي يحتاجها. وعند الخروج من هذه الحلقة يتم الرد على مدير الموارد بشعاع يبين الحاويات التي تم اسنادها إلى الطلبات وبعدها يتم فحص وجود أعمال جديدة ليتم إضافتها إلى الرتل على شكل طلبات حجز لتطبيقات رئيسية.

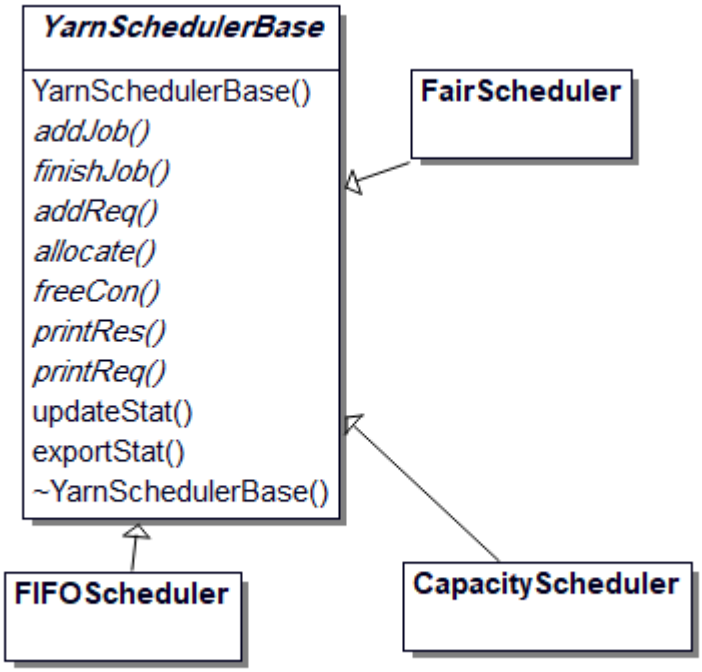
2– المجدول من نوع FIFO الشكل (4–9): يتم اختبار وجود طلبات في الرتل كما يتم اختبار وجود حاويات فارغة بعدها يتم استخراج أول طلب من الرتل وهنا نميز حالتين:

a. عدد الحاويات الفارغة أكبر من الحاويات التي يحتاجها الطلب، بالتالي يتم إسناد كل ما يحتاجه الطلب وإنقاص عدد الحاويات الفارغة بنفس المقدار ومن ثم يتم حذف الطلب. يتم تكرا ذلك حتى يصبح الرتل فارغا أو عدد الحاويات الفارغة مساويا للصفر عندها يتم الرد على مدير الموارد بشعاع يبين الحاويات التي تم اسنادها إلى الطلبات وبعدها يتم فحص وجود أعمال جديدة ليتم إضافتها إلى الرتل على شكل طلبات حجز لتطبيقات رئيسية.

b. عدد الحاويات الفارغة أصغر من عدد الحاويات التي يحتاجها الطلب بالتالي يتم إسناد كل الحاويات الفارغة إلى الطب ومن ثم يتم تصفير عدد الحاويات الفارغة. يتم الرد على مدير الموارد بشعاع يبين الحاويات التي تم اسنادها إلى الطلبات وبعدها يتم فحص وجود أعمال جديدة ليتم إضافتها إلى الرتل على شكل طلبات حجز لتطبيقات رئيسية.

3– المجدول من نوع Capacity الشكل (4–11): إن هذا المجدول هو عبارة عن مجموعة من الأرتال من نوع FIFO وكل رتل مميز باسم وبنسبة مئوية من الموارد. يقوم المجدول السعوي ومن اجل عملية حجز الموارد باستدعاء الدالة allocate من اجل كل الارتال التي يحتويها. وعند ورود عمل جديد يتم اضافة العمل إلى رتل معين بناء على الحقل Qname في العمل والذي يحتوي على اسم الرتل.

الشكل التالي يبين مخطط UML لأنماط الجدولة الثلاثة حيث أنها ترث الصف YarnSchedulerBase

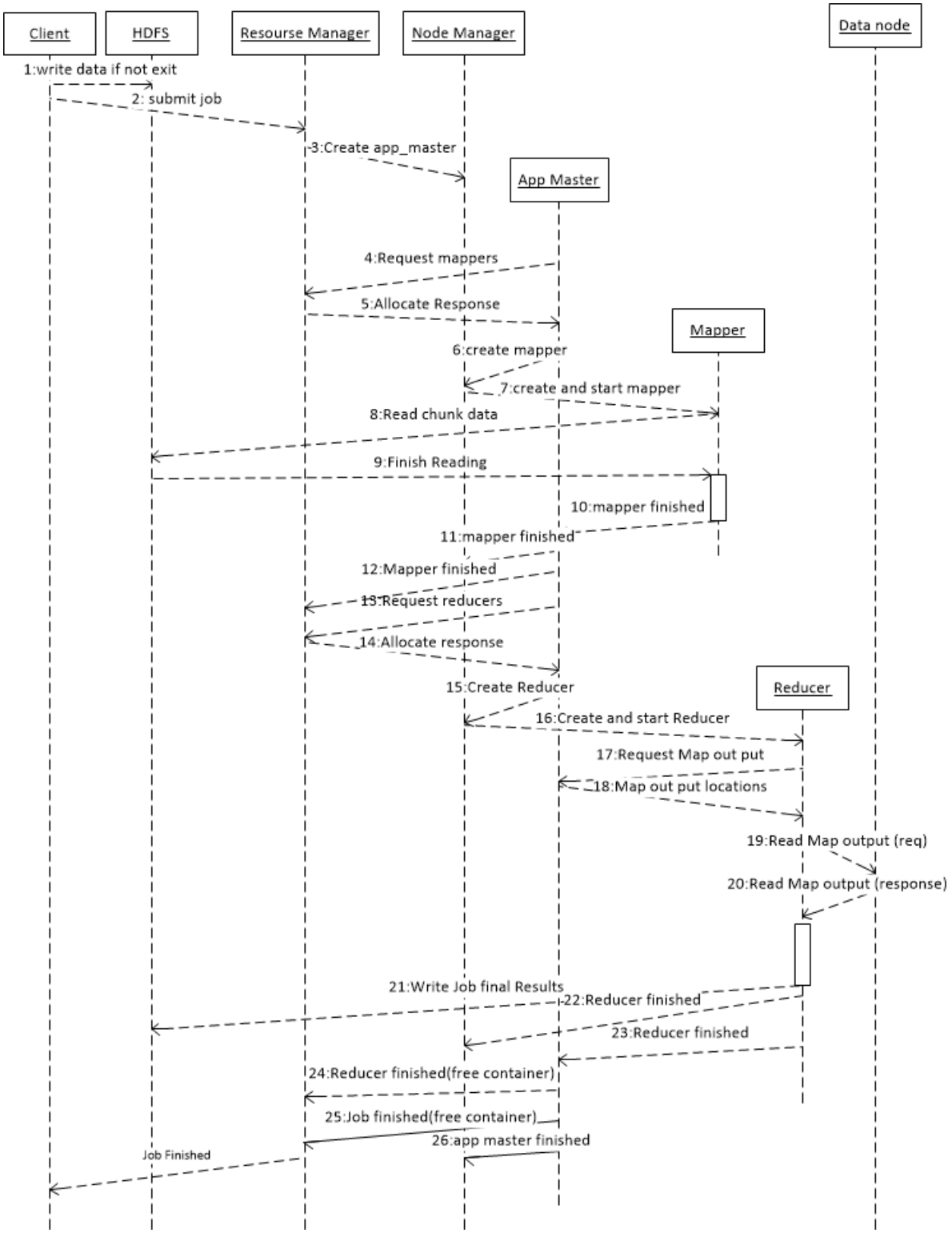


الشكل (4–12) مخطط UML يمثل أنماط الجدولة التي ترث الصف YarnSchedulerBase

4–3–3–4 تحقيق المقابلة والاختزال (Map Reduce)

إن تحقيق المقابلة والاختزال يعتمد بشكل أساسي على تحقيق YARN وعلى تحقيق نظام الملفات الموزع وقد حاولنا محاكاة معظم الرسائل بحيث نصل إلى أكبر دقة ممكنة و من الرسائل التي لم نتطرق اليها رسائل تحديث الحالة و التي تعبر عن نسبة الانتهاء من المهام و الأعمال.

يبين لنا الشكل (4–13) تسلسل تنفيذ عمل المقابلة والاختزال ويمكن تلخيص ذلك بالخطوات التالية:



الشكل (4–13) مخطط UML تسلسلي يوضح آلية تنفيذ أعمال المقابلة والاختزال

1– يجب أن يقوم الزبون بكتابة الملفات اللازمة للعمل إن لم تكن موجودة مسبقا وطبعا هنا سيستخدم الزبون الصف HDFSClient.

2– بعد ذلك يقوم الزبون بإرسال العمل إلى مدير الموارد لكي يتم تنفيذه.

3- يقوم مدير الموارد بإرسال رسالة إلى مدير عقدة (يتم تحديده من قبل المجدول) بحيث يقوم بحجز حاوية تمثل التطبيق الرئيسي لعمل المقابلة والاختزال.

4- بعد أن يتم إنشاء التطبيق الرئيسي يقوم الأخير بإرسال رسالة إلى مدير الموارد من أجل حجز حاويات ليتم تنفيذ المقابلات عليها (Mappers).

5- يقوم مدير الموارد بإرسال رد على طلب حجز المقابلات وهو عبارة عن شعاع يحتوي على الأجزاء (Chunks) والحاويات التي سيتم عليها تنفيذ الأجزاء علما أنه يتم حجز الموارد إما دفعة واحدة أو على دفعات وذلك بحسب توفر الموارد.

6- بعدها يقوم التطبيق الرئيسي بإرسال الرد على حجز الموارد إلى مدير العقدة التي تحتوي على الحاوية.

7- يقوم مدير العقدة بإنشاء المقابل.

8- يقوم المقابل بإرسال طلب قراءة إلى نظام الملفات الموزع من أجل قراءة الجزء الذي سيتم التنفيذ عليه.

9- يقوم نظام الملفات الموزع بإرسال إشعار يبين الانتهاء من عملية قراءة الجزء.

10- يقوم المقابل بتنفيذ تابع المقابلة وعند الانتهاء يقوم بإرسال رسالة الانتهاء من التنفيذ إلى التطبيق الرئيسي الذي يزيد عدد المقابلات التي انتهت من التنفيذ ويحفظ مكان تخزين خرج المقابلات.

11- يقوم التطبيق الرئيسي بإعلام مدير العقدة بأن المقابل قد انتهى من عمله.

12- يقوم التطبيق الرئيسي بإعلام مدير الموارد بأن المقابل قد انتهى من عمله بحيث يتم تحرير الحاوية.

13- في حال كان عدد المقابلات التي انهت عملها أكبر من المعامل Map-Reduce job.reduce.slowstart.completedmaps. يقوم التطبيق الرئيسي بإرسال طلب حجز المختزلات إلى مدير الموارد.

14- عند توفر الموارد يقوم مدير الموارد بإرسال رد على طلب حجز المختزلات على شكل شعاع يحتوي رقم المختزل والحاوية التي سيتم تنفيذه عليها.

15- يقوم التطبيق الرئيسي بإرسال الرد على حجز المختزل إلى مدير العقدة التي تحتوي الحاوية التي سيتم تنفيذ المختزل عليها.

16- يقوم مدير العقدة بإنشاء المختزل.

17- يقوم المختزل بإرسال رسالة إلى التطبيق الرئيسي يطلب من خلالها أماكن خرج المقابلات.

18- يقوم التطبيق الرئيسي بإرسال أماكن خرج المقابلات إلى المختزل (يمكن أن ترسل أماكن خرج المقابلات إما دفعة واحدة أو على دفعات وذلك يعتمد على تسلسل التنفيذ)

19- بعدها يقوم المختزل بإرسال طلب قراءة خرج المقابل من عقدة البيانات التي تحتوي على المقابل (عمليا القراءة هنا من نظام الملفات المحلي لعقدة البيانات).

20- عند الانتهاء من قراءة خرج المقابل تقوم عقدة البيانات بإعلام المختزل بذلك.

21- يقوم المختزل بتنفيذ تابع الاختزال ويكتب الخرج على نظام الملفات الموزع.

22- عندما ينتهي المختزل يقوم بإعلام التطبيق الرئيسي بذلك.

23- عندما ينتهي المختزل يقوم بإعلام مدير العقدة بذلك.

24- يقوم التطبيق الرئيسي بإعلام مدير الموارد بانتهاء المختزل لكي يتم تحرير الموارد التي حجزت له.

25- في حال انتهاء كل المختزلات يكون العمل قد تم تنفيذه بالتالي يقوم التطبيق الرئيسي بإعلام مدير الموارد بذلك.

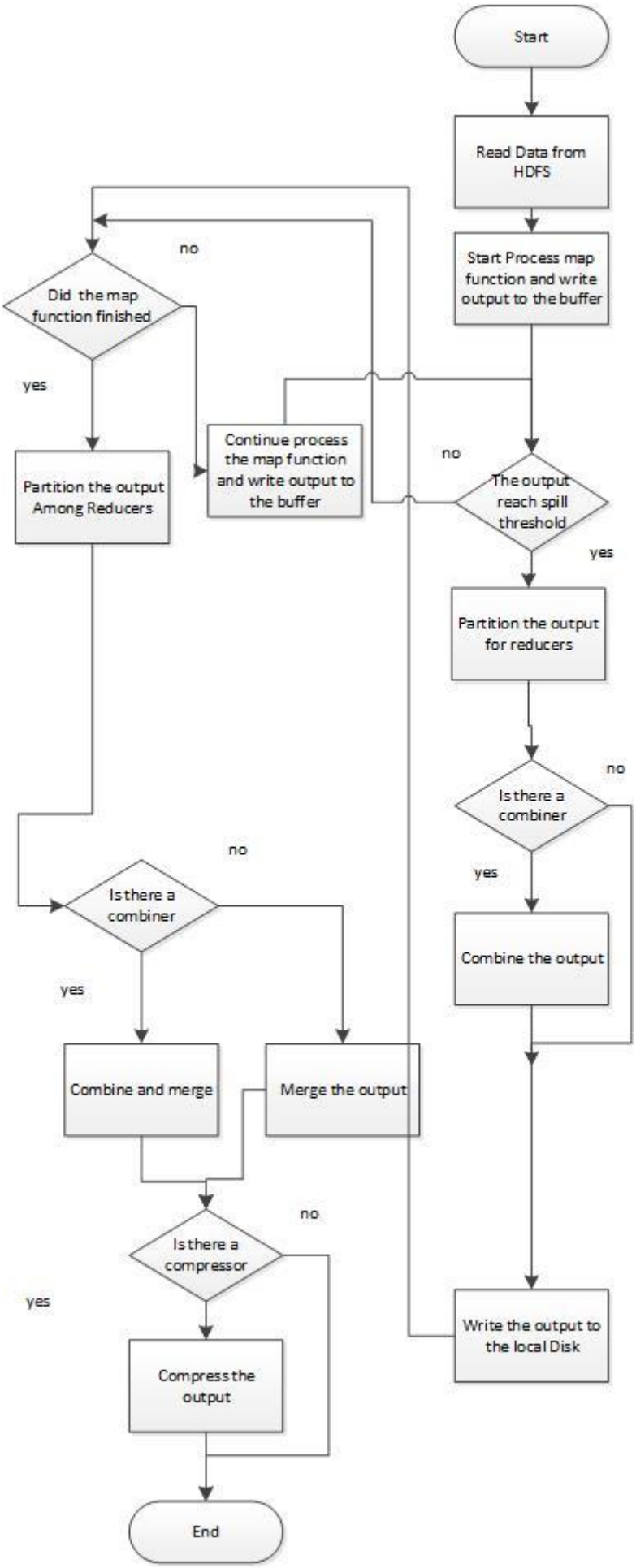
26- كما يقوم التطبيق الرئيسي بإعلام مدير العقدة التي تحتويه بأن التطبيق الرئيسي انهى مهمته.

27- يقوم مدير الموارد بإعلام الزبون بانتهاء العمل.

4-3-3-4-1 تحقيق المقابل Mapper

إن تنفيذ تابع المقابلة ينطوي على عدد من الخطوات الموضحة بالشكل (4-14) فبعد أن يتم إنشاء المقابل ومن أجل كل جزء من أجزاء البيانات، يقوم المقابلة بقراءة هذا الجزء من نظام الملفات الموزع، وبعدها يقوم بتنفيذ تابع المقابلة على هذا الجزء ونتيجة التنفيذ يتم تخزينها بالذاكرة حتى يصل حجم الخرج إلى عتبة التفريغ، عندما يصل حجم الخرج في الذاكرة أكبر من عتبة التفريغ يبدأ التفريغ على القرص الصلب المحلي وفي حال امتلاء العازل الذاكري أثناء عملية التفريغ يتوقف تنفيذ المقابل حتى الانتهاء من تفريغ كامل العازل إلى القرص الصلب. في حال وجود مجمع (Combiner) يتم استدعائه على البيانات قبل تفريغها من أجل تقليل حجم البيانات المراد كتابتها على القرص الصلب. وعند انتهاء المقابل من التنفيذ تبدأ مرحلة الدمج، ففي البداية يتم استدعاء المقسم من أجل تقسيم الخرج على كل

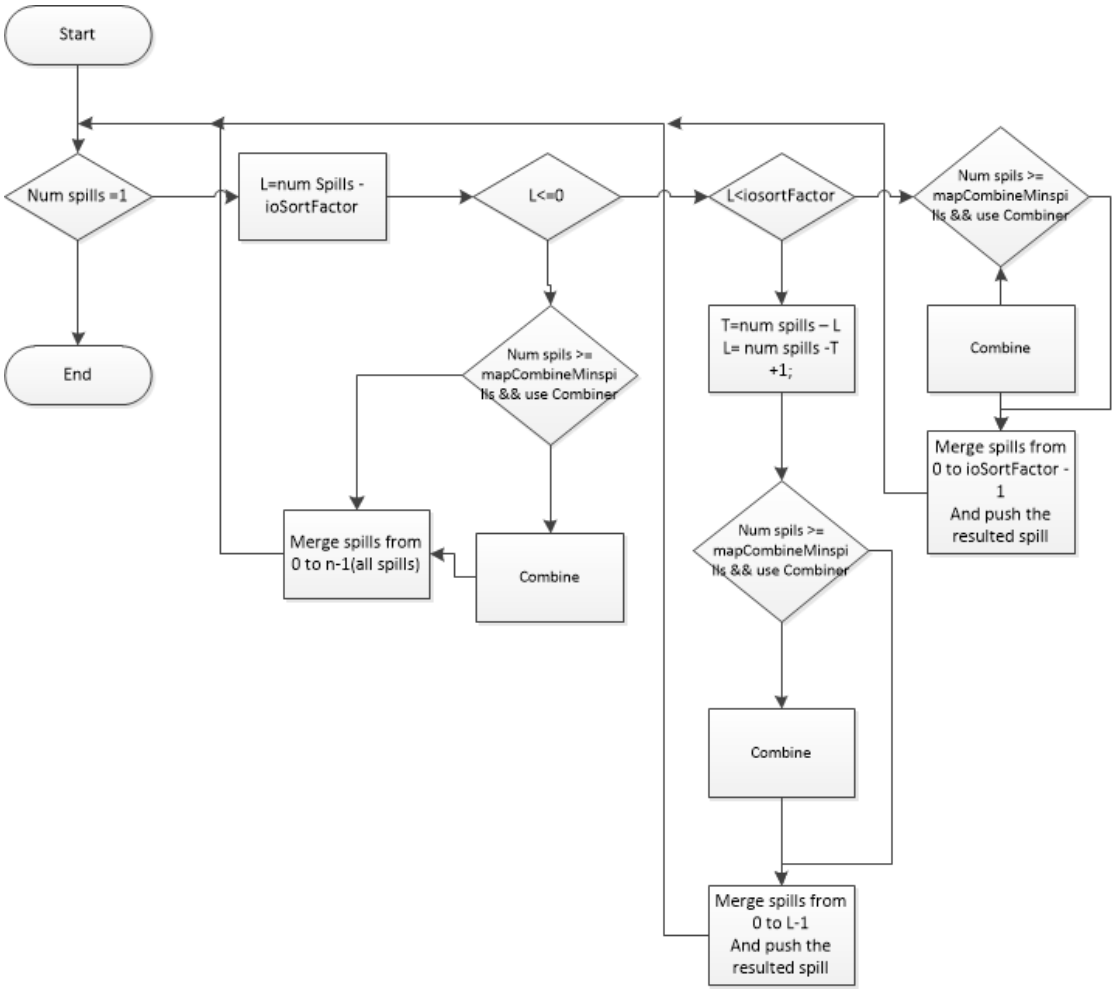
المختزلات بعدها يتم الدمج وفي حال وجود مجمع يتم استدعائه ثم يتم الدمج بعد ذلك، في حال وجود ضاغط يتم ضغط الخرج ومن ثم يتم تخزينه على القرص الصلب.


Start
Read Data from HDFS
Start Process map function and write output to the buffer
Did the map function finished
no
yes
Continue process the map function and write output to the buffer
Partition the output Among Reducers
The output reach spill threshold
no
yes
Partition the output for reducers
Is there a combiner
no
yes
Combine the output
Is there a combiner
no
yes
Combine and merge
Merge the output
Is there a compressor
no
yes
Compress the output
Write the output to the local Disk
End

الشكل (4–14) مخطط تدفقي يوضح آلية تنفيذ مهمة المقابلة

4–3–3–4–2 تحقيق الدامج Merger

يتم استخدام الدامج من قبل المقابل ومن قبل المختزل حيث يقوم بدمج ملفات التفريغ بحيث تصبح ملف. إن خوارزمية الدامج تقوم على قراءة أكثر من ملف ودمجهم في ملف واحد بشرط أن تكون أخر موجة للدمج تحتوي على عدد من الملفات مساويا للمعامل io sort factor كما أن عدد الملفات الداخلة في أي موجة من موجات الدخل يجب ألا يتجاوز هذا المعامل. وفي الشكل(4–15) مخطط تدفقي يوضح الآلية المتبعة لتحقيق الدامج.



الشكل(4–15) آلية عمل الدامج

4–3–3–4–3 تحقيق المجمع Combiner

إن المجمع يقوم بعمل مشابه لتابع الاختزال فهو يجمع السجلات المشتركة بالمفتاح نفسه فمثلا ومن أجل تطبيق عداد الكلمات يكون خرج المقابل بالشكل التالي.

```
<What,1> <do,1> <you,1> <mean,1> <by,1> <Object,1>
<What,1> <do,1> <you,1> <know,1> <about,1> <Java,1>
<What,1> <is,1> <Java,1> <Virtual,1> <Machine,1>
<How,1> <Java,1> <enabled,1> <High,1> <Performance,1>
```

و عند إدخال هذه السجلات على المجمع سينتج

```
<What,3> <do,2> <you,2> <mean,1> <by,1> <Object,1>
<know,1> <about,1> <Java,3>
<is,1> <Virtual,1> <Machine,1>
<How,1> <enabled,1> <High,1> <Performance,1>
```

نلاحظ أن عدد السجلات تم اختزاله كما أن طول السجل قد اختلف بالتالي هذه هي القيم التي تهمنا عند محاكاة المقابلة والاختزال. ومن أجل توقع عدد السجلات الناتجة عن المجمع قدم S.Hammoud العلاقة (4–1)[18] حيث درس عدد السجلات الناتجة عن هذه العلاقة وقارنها مع التجارب الواقعية، فكانت نسبة الخطأ لا تتجاوز 5% وهي نسبة قليلة جدا، بالتالي قمنا باستخدام هذه العلاقة لتحقيق المجمع

$$f(n) = \frac{1 - a^n}{1 - a}, \qquad a = \frac{g - 1}{g}$$

العلاقة (4–1) استنتاج عدد السجلات الناتجة عن المجمع [18]

حيث:

- F(n): عدد السجلات الناتجة عن إدخال n سجل إلى المجمع
- g: عدد المفاتيح الكلي.

قام S.Hammod باستخدام الحدث المتمم لبرهان العلاقة السابقة وقد قمنا في هذا البحث بمحاولة ايجاد علاقة أخرى لاستنتاج عدد السجلات حيث افترضنا التالي: إن المجمع يقوم بدمج السجلات التي تمتلك نفس المفتاح فهو عبارة عن تابع F(n,g) دخله عدد السجلات الكلي n وg هو عدد المفاتيح الكلي وخرجه عدد من السجلات الفريدة المختلفة عن بعضها البعض. ولنفترض أن احتمالات ظهور أي مفتاح في أي سجل هي احتمالات متساوية. فمن أجل السجل الأول يكون احتمال ان يكون ينتمي إلى المفتاح الأول هو 1/g وأن ينتمي إلى المفتاح الثاني 1/g وهكذا بالنسبة لباقي المفاتيح. بالتالي عدد الإمكانيات الكلي هو g*g*g...*g أي g^n من أجل توقع عدد السجلات الناتج .

ليكن T(n,1) هو عدد الإمكانات التي يكون فيها n سجل تنتمي لنفس المفتاح أي أن كل سجل له حالة واحدة وهذه الحالة يتم اختيارها من g حالة أي c(g,1).

$$T(n,1) = C(g,1)*1n$$

T(n,2) هو عدد الإمكانات التي يكون فيها n سجل تنتمي لفتاحين مختلفين, أي كل سجل له حالتين كما يجب طرح الحالات التي تكون فيها كل السجلات تنتمي لنفس المفتاح أي:

$$T(n,2)= C(g,2)*(2n - C(2,1)*(1n))$$

T(n,3) هو عدد الإمكانات التي يكون فيها n سجل تنتمي لثلاث مفاتيح مختلفة، أي أن كل سجل له ثلاث حالات مختلفة وهذه الحالات يتم اختياره من g حالة أي C(g,3)، كما يتم طرح الحالات التي تكون فيها السجلات من نوعين مختلفين والحالات التي تكون فيها السجلات من نوع واحد.

$$T(n,3) = C(g,3)*(3n-[C(3,2)* [2n - C(2,1)*1n]] - C(3,1)*1n)$$

و هكذا حتى الوصول إلى T(n,g) وهو عدد الامكانيات التي يكون فيها n سجل تنتمي إلى g مفتاح مختلف. عمليا كل احتمال يشمل احتمال ورود كل المفاتيح في كل سجل مطروحا منه احتمال غياب أحد هذه المفاتيح أو أكثر. ويمكن تعميم العلاقات السابقة بالشكل التالي:

فمن أجل n سجل وg مفتاح يكون عدد الإمكانيات لظهور جميع هذه المفاتيح في هذه السجلات هو

$$T(n,g) = C(n,g) * \sum_{j=1}^{g-1} T(n,j)$$

و باعتبار:

$$T(n,0)=0$$

فعلى سبيل المثال ومن أجل 5 مفاتيح و5 سجلات ستكون لدينا الإمكانات التالية

k	عدد امكانات ظهور k مفتاح	P(k) الاحتمال	P(k)*k
1	5	5/3125	0.0016
2	300	300/3125	0.192
3	1500	1500/3125	1.44
4	1200	1200/3125	1.536
5	120	120/3125	0.192
المجموع الكلي	=312555	1	3.366

إن الأعداد العلاقة السابقة يمكن تمثيلها بعلاقة مختصرة [42]

$$T(n,g) = C(n,g) * \sum_{n}^{i=g} -1^{g-1} * C(g,i) * j^{n}$$

إن هذه السلسلة وعند ضرب كل عنصر من عناصرها بعدد المفاتيح وجمع القيم وتقسيم الناتج على عدد الإمكانات الكلي سنحصل على عدد المفاتيح الفريدة المتوقع.

أي أن

$$Estimated\ keys = \sum_{1}^{g} \frac{T(n,g) * j}{g * n} = \frac{E(n,g)}{g * n}$$

أثبت Thomas Dybdahl Ahle, [43] أن:

$$E(n,g) = g * (g^{n} - (n-1)^{n})$$

بالتالي سيكون عدد المفاتيح الفريدة المتوقع هو:

$$Estimated\ keys = \frac{g * (g^n - (n-1)^n)}{g * n} = g - g * (\frac{g-1}{g})^g$$

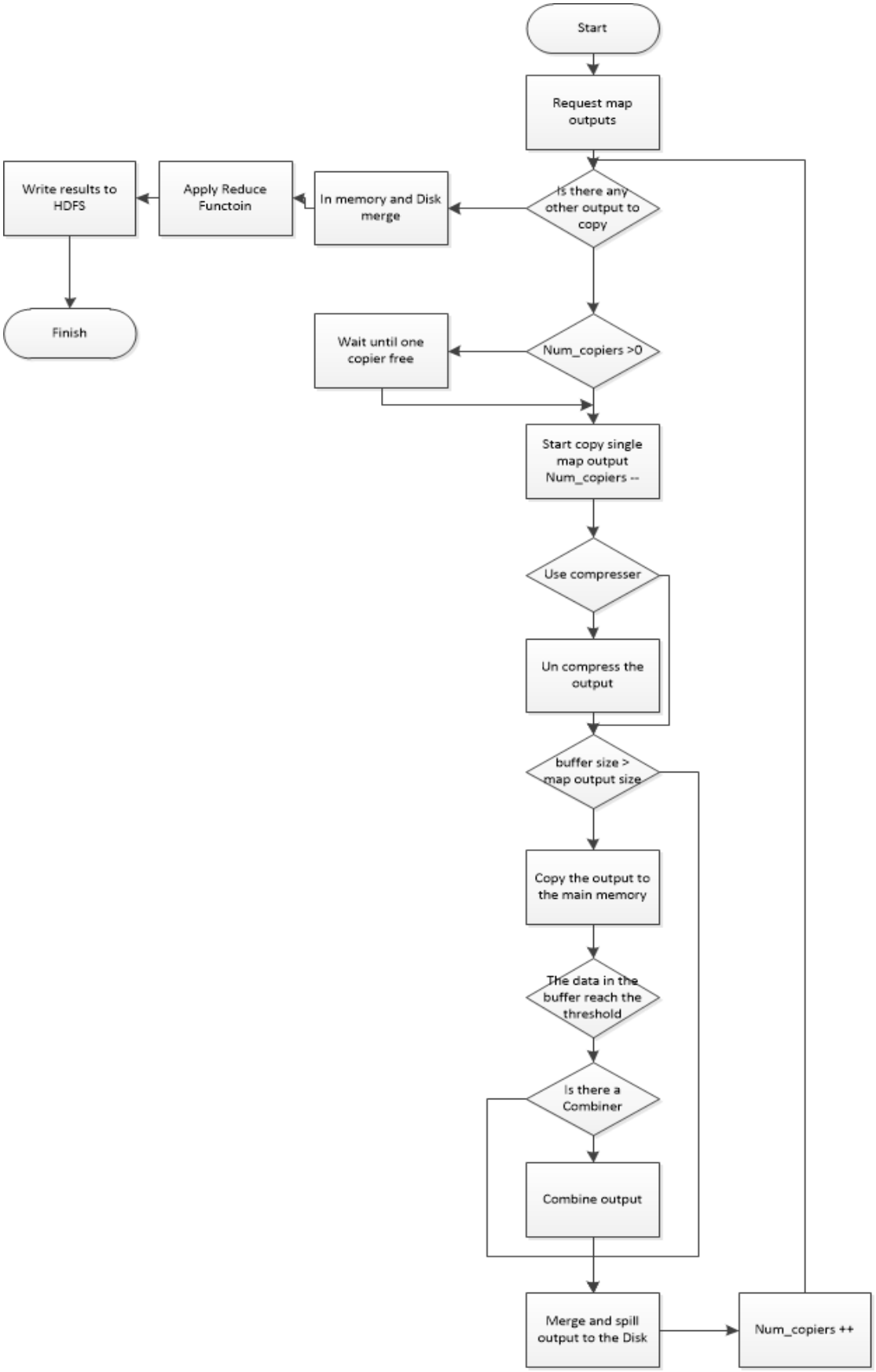
والعلاقة الأخيرة مطابقة للعلاقة (4–1) المقترحة من قبل S.Hammod أي أننا قمنا ببرهان العلاقة بأسلوب ثان ولم نتوصل لعلاقة جديدة.

4–3–3–4–4 تحقيق المختزل (Reducer)

إن مرحلة الاختزال هي المرحلة الثانية والأخيرة من تطبيق المقابلة والاختزال. وهي تنطوي على عدد من الخطوات التي قمنا بتحقيقها في هذا المحاكي وهي موضحة بالمخطط التدفقي (4–16).

فحالما يتم إنشاء المختزل يقوم بإرسال طلب إلى التطبيق الرئيسي من أجل الحصول على أماكن خرج المقابلات. ويمكن أن يصله مكان خرج المقابلات على دفعات أو على دفعة واحدة، يقوم المختزل بنسخ خرج المقابلات حالما يتوفر مع مراعات عدد المسالك المسؤولة عن النسخ وهي بشكل افتراضي 5، وفي حال كانت المسالك الخمسة مشغولة يتم الانتظار حتى يصبح أحد المسالك متاحا، وفي حال كان هنالك ضغط للبيانات فيجب أن يتم فك ضغط خرج المقابلات الذي تم نسخه، بعدها وفي حال كان حجم خرج المقابل المنسوخ أكبر من العازل الذاكري يتم دمجه وتفريغه بشكل مباشر على القرص الصلب، وفي حال العكس يتم كتابة الخرج على الذاكرة الرئيسية وفي حال وصل حجم البيانات المكتوبة على الذاكرة إلى عتبة التفريغ يتم التفريغ على القرص.

عندما يتم نسخ خرج كل المقابلات يقوم المختزل بدمج ملفات التفريغ التي على القرص مع البيانات التي على الذاكرة ومن ثم يقوم بتطبيق تابع الاختزال والنتائج يتم كتابتها على نظام الملفات الموزع وهكذا يكون المختزل قد أنها عمله.



الشكل (4–16) آلية عمل المختزل

4–3–3–4–5 تحقيق آلية إدخال بيانات أعمال المستخدم

إرسال عمل إلى المحاكي

من أجل إرسال عمل إلى المحاكي يتم إنشاء ملف من نوع json وفق الشكل (4–17) ويتم وضعه في المسار الذي يحوي المحاكي وضمن المجلد resources\jobs. كما يمكن أن نضع عدة ملفات من أجل إرسال عدة أعمال إلى المحاكي في نفس الوقت.

```
{       "jobName":"job",                                  اسم العمل
        "numberOfReducers":1,                             عدد المختزلات
        "useCombiner":false,                              استخدام مجمع ام لا
        "useCompression":true,                            استخدام الضغط أم لا
        "numOfFiles":1,                                   عدد ملفات الدخل
        "maxFileSize":3000000000,                         أقل حجم للملف Byte
        "minFileSize":3000000000,                         أكبر حجم للملفByte

        "mergeCost":10,                                   كلفة عملية الدمج من أجل كل
                                                          سجل(FP)
        "mapCost":33,                                     كلفة عملية المقابلة من أجل كل
                                                          سجلFP
        "recordSize":295,                                 حجم سجل الدخل للمقابل Byte
        "mapOutRecords":50,                               عدد سجلات خرج المقابل بالنسبة
                                                          لكل سجل دخلByte
        "mapOutAvRecordSize":12,                          معدل حجم سجل خرج المقابل
                                                          byte

        "reduceCost":12,                                  كلفة المختزل عملية الاختزالFp
        "reduceRecords":0.1,                              نسبة سجلات خرج المختزل بالنسبة
                                                          لدخله
        "reduceOutAvRecordSize":10,                       معدل حجم سجل خرج المختزل
                                                          Byte
```

"compressionCost":1.0,	كلفة الضغط بالنسبة لكل سجل Fp
"uncompressionCost":1.0,	كلفة عملية فك الضغط Fp
"compressionSize":0.5 ,	نسبة الحجم بعد الضغط
"combineCost":1,	كلفة المجمع من أجل كل سجل Fp
"combineGroups":100000,	عدد المفاتيح بالنسبة لبيانات الدخل
"combineOutAvRecordSize":20,	معدل حجم السجل الناتج عن المجمع byte
"combinerType":"eq",	نوع المجمع eq/percent
"combineRecordesPercent":0.01,	
"Map–Reduce ParallelCopies":5,	عدد عمليات النسخ الأعظمي التي يمكن أن تعمل في نفس الوقت خلال مرحلة الاختزال
"ioSortMb":20.0,	حجم العازل الذاكري المخصص لعملية الترتيب (خاص بالمقابل)
"ioSortSpillPercent":0.80,	نسبة من العازل إذا تم اشغالها تبدأ عملية التفريغ على القرص
"ioSortFactor":10,	عدد ملفات الدمج الأعظمي في كل موجة من موجات الدمج
"mapredJobReduceInputBufferPercent":0.7,	نسبة من حجم العازل الذاكري إذا كان خرج المقابل أكبر منها تتم الكتابة مباشرة على القرص الصلب (خاص بالمختزل)
"mapredJobShuffleMergePercent":0.66,	نسبة من حجم العازل الذاكري إذا انشغلت تبدأ بعدها عملية التفريغ على القرص (خاص بالمختزل)
"memoryLimit":100000000.0	حجم العازل الذاكري المخصص للمختزل Byte
"queueName":"q1"	اسم الرتل ويستخدم في حالة المجدول السعوي لتحديد الرتل الذي سيتم تنفيذ العمل عليه
}	

الشكل (4–17) ملف json يمثل عمل مقابلة واختزال

إعداد العنقود

من أجل إعداد العنقود الحاسوبي يتم استخدام ملفين من نوع XML هما:

- hsgPlatform.xml يمثل بيئة المحاكاة حيث يحدد نوع الأجهزة والوصلات والأقراص الصلبة والمعالجات.
- hsgDeploy.xml ويقوم بنشر العقد (عقد البيانات وعقدة الأسماء ومدير الموارد ....) على الأجهزة ضمن العنقود.

هذان الملفان موجودان في مسار المحاكي ضمن المجلد resources. ومن أجل تسهيل عملية إعداد التجارب قمنا بتحقيق وحدة مسؤولة عن التوليد الآلي لملفات البيئة والنشر وفق خصائص يتم تحديدها في الملف cluster.json الموجود في المسار resources\cluster والشكل (4–18) يبين مثالا عن هذا الملف.

```
{
    "generatePlatformAndDeploy":true,        من أجل توليد أو عدم توليد ملفات بيئة ونشر جديدين
    "racksNum":3,                            عدد الرفوف
    "hostsPerRack":10,                       عدد الأجهزة ضمن الرف الواحد
    "cpuSpeed":"2.0Gf",                      سرعة المعالج
    "coresNum":"4",                          عدد النوى لكل معالج
    "hddNums":3,                             عدد الأقراص الصلبة
    "hddWriteSpeed":"130MBps",               سرعة الكتابة على للقرص الصلب
    "hddReadSpeed":"130MBps",                سرعة القراءة من القرص الصلب
    "hddreadAccess":0.020,                   زمن الوصول، عند القراءة بالثانية
    "hddwriteAccess":0.020,                  زمن الوصول عند الكتابة بالثانية
    "hddSlice":0.087,                        الشريحة الزمنية لعملية القراءة/الكتابة في حال وجود أكثر من عملية

    "accessLinkSpeed":"1GBps",               سرعة الوصلة بين كل جهاز (مضيف)و الموجه ضمن نفس الرف
    "accessLinkLatency":"50us",              التأخير الشبكي على وصلة مضيف– موجه ضمن نفس الرف
```

"backBoneLinkSpeed":"5GBps",	سرعة الوصلة التي تمثل الموجه
"backBoneLinkLatency":"100us",	التأخير ضمن الموجه
"coreLinkSpeed":"5GBps",	سرعة الوصلة التي تصل بين الموجهات الشبكية التي تربط أكثر من رف
"coreLinkLatency":"100us",	التأخير الشبكي على الوصلات التي تربط الرفوف
"SchedulerType":"fair",	نمط الجدولة FIFO/fair/capacity
"chunkSize":128,	حجم الجزء MB (خاص ب HDFS)
"replicatinNum":3,	معامل تكرار البيانات (HDFS)
"slowStartNumFinishedMappers":5.0,	النسبة المئوية للمقابلات التي انتهت من عملها وبعدها يتم طلب حجز المختزلات
"numCorePerContainer":1	عدد المعالجات المحددة لكل حاوية
}	

الشكل (4–18) الملف cluster.json والذي يحدد خصائص العنقود الحاسوبي

في حال استخدام المجدول السعوي (capacity) يجب ان يحتوي المجلد resources\cluster على ملف capacity.json يحدد فيه أسماء الأرتال ونسبة الموارد المحجوزة لكل رتل من اجمالي الموارد في العنقود والشكل (4–19) يحتوي على مثال لملف capacity.json.

```
{"capacity":[
{ "name":"q1",
 "percent":5
},
{
"name":"q2",
"percent":95
}]}
```

الشكل (4–19) نموذج عن ملف إعداد المجدول السعوي

بعد الانتهاء من تنفيذ أي عمل من أعمال المقابلة والاختزال يتم توليد ملف للنتائج موافق لاسم العمل متبوعا بتاريخ التنفيذ وذلك ضمن المجلد resources\results والشكل (4–20) يبين لنا جزء من

ملف النتائج. كما يتم توليد ملف يبين احصائيات رتل المجدول وذلك ضمن المسار resources\results\q وفي حالة المجدول السعوي يتم توليد ملف لكل رتل وفق الشكل (4–21).

job name	wc 1000 0
MAP_INPUT_RECORDS	10485760.000000
MAP_OUTPUT_RECORDS	104857599.562500
MAP_INPUT_SIZE	1048576000.000000
MAP_OUTPUT_SIZE	1677721593.000000
COMBINE_INPUT_RECORDS	0.000000
COMBINE_OUTPUT_RECORDS	0.000000
REDUCE_INPUT_RECORDS	104857593.000000
SPILLED_RECORDS	638249813.000000
REDUCE_SHUFFLE_BYTES	1677721488.000000
JOB_START_TIME	101.208560
JOB_STOP_TIME	867.396027
JOB_TOTAL_TIME	766.187468
SHUFFLE	375.480895
avMappersTime	26.673259
avReducersTime	320.858135
Data_local_map_tasks	18.000000

الشكل (4–20) مثال عن ملف نتائج أحد أعمال المقابلة والاختزال.

Statistic name	Value
Average delay	17.930182
Average Q length	2.035851
Cluster utilization	0.062645

الشكل (4–21) مثال عن احصائيات الارتال

و للاطلاع على كامل النص البرمجي للمحاكي HSG فإنه متوفر على الرابط[1]

[1] https://github.com/amerdwar/hsg2/

4-4 الخاتمة

انطلقنا في هذا الفصل من المتطلبات الواجب تحقيقها في المحاكي HSG، إن أهم هذه المتطلبات هي اختيار المحاكي الشبكي حيث وقع اختيارنا على المحاكي SimGrid، إن SimGrid يعتبر مناسبا للتطبيقات الموزعة واسعة النطاق. قدمنا بعدها التصميم العام للمحاكي HSG ثم قمنا بتفصيل كل مكون على حدا، كما دعمنا مكونات المحاكي بعدد من المخططات التسلسلية التي تبين أسلوب عمل هذا المحاكي.

الفصل الخامس

الاختبارات العملية

5–1 المقدمة

قمنا في الفصل السابق بتقديم المحاكي HSG سنقوم في هذا الفصل بالتحقق من دقة نتائجه ومقارنة أدائه مع عدد من المحاكيات الأخرى مبينين الميزات التي يتمتع بها مقارنة بغيره من المحاكيات.

5–2 اختبار نظام ملفات Hadoop الموزع (HDFS)

لاختبار النموذج المقترح لـHDFS أجرينا عدد من التجارب تبعاً لعدد من المعاملات ومقارنة نتائج المحاكاة مع النتائج الواقعية التي أجراها F.Tian وآخرون[44]، حيث اختبروا أداء HDFS على عنقود مؤلف من ثمان عقد بيانات وعقدة Name Node والجدول(5–1)يبين المواصفات الفنية لهذه العقد.

Type	NameNode	DataNode
CPU	2*4 core Intel Xeon CPU (2.00 GHz)	4 core Intel Core i5 CPU (2.67 GHz)
Memory	8 GB	8 GB
Hard drive	2*146 GB Seagate SAS hard drive (15,000 rpm)	500 GB Seagate SATA hard drive (to install OS and software, 7200 rpm) 1 TB Seagate SATA hard drive (to store the data of HDFS, 7200 rpm)

الجدول (5–1) مواصفات العقد التي استخدمت في المرجع [44] من أجل الاختبارات الواقعية على HDFS

إن سرعة القراءة والكتابة للقرص الصلب الخاص بعقدة الاسماء تتراوح بين 171–110 MB/s [45]

و بالنسبة للقرص الخاص بعقد البيانات فإن معدل النقل فيه 210 MB/s [46]

تم ربط هذه العقد باستخدام وصلاتGigaByte Ethernet والمحول الشبكي من نوع TP–Link TL–SG2248WEB.

5–2–1 اختبار الكتابة

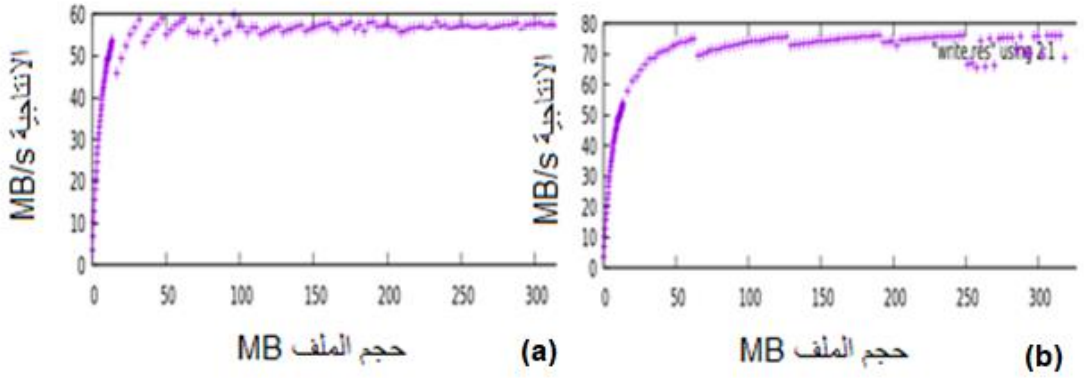
كتب الباحثون 78 مجموعة من البيانات إذ كل مجموعة تتألف من 500 ملفاً لها الحجم نفسه، حيث تم قياس زمن الكتابة لكل ملف ومن ثم تم أخذ وسيط الأزمنة في كل مجموعة. هذه المجموعات تتزايد ملفاتها بالحجم من 0.25 إلى 320 MB، وتم إعادة التجربة مع تغيير حجم ال Chunk (16,64,96,128) وذلك لدراسة تأثير حجم الـChunk على أداء HDFSوباعتبار معامل التكرار للبيانات 3.

في هذا البحث اعتمدنا على اختبارات الباحثين في [44] من اجل التحقق من أداء HDFS ضمن المحاكي HSG .أعددنا عنقود حاسوبي يحتوي 8 عقد بيانات وعقدة NameNode بحيث تكون مواصفات هذه العقد (الأقراص الصلبة والمعالجات) والوصلات والمبدلات الشبكية مشابهة للمستخدمة في [44] . الشكل (1–5) يبين لنا مواصفات العقد في المحاكي وسرعة الوصلة.

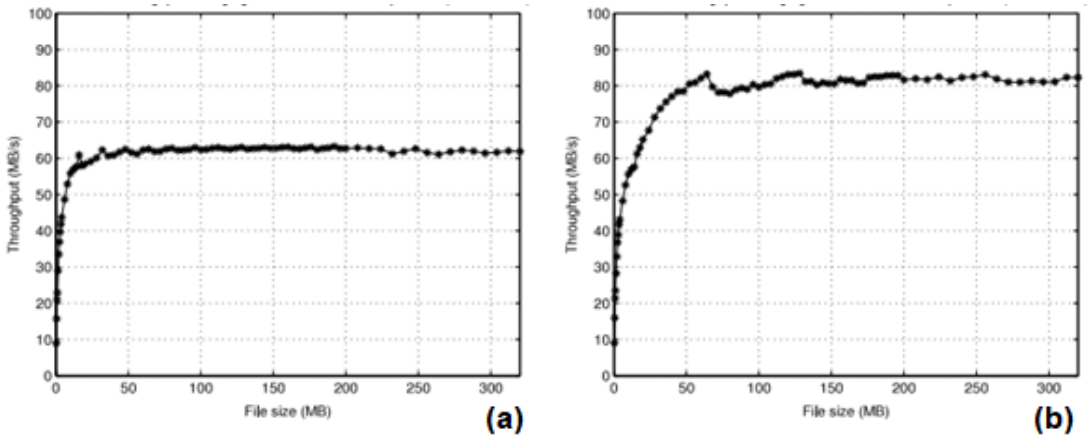
```
<host id="host9" core="4" speed="2.67Gf">
    <mount name="/h9d1" storageId="host9_disk1"/>
</host>
<link id="l61" latency="50us" bandwidth="1GBps"/>
```

كررنا الاختبار من أجل حجوم مختلفة لـ (Chunk) ورمزنا لحجم الـChunk بـ BS16 اذا كان حجمه 16 MB و الشيء نفسه بالنسبة لباقي الحجوم.

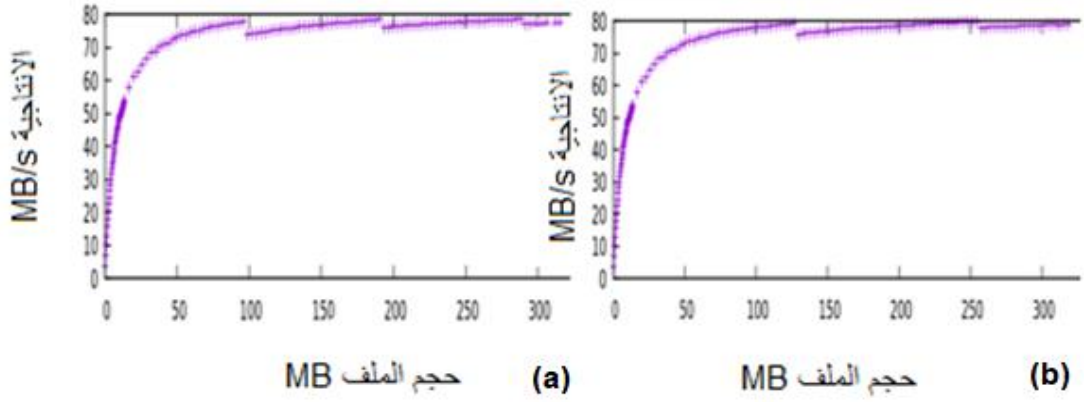
وكانت نتائج أداء الكتابة على HDFS:



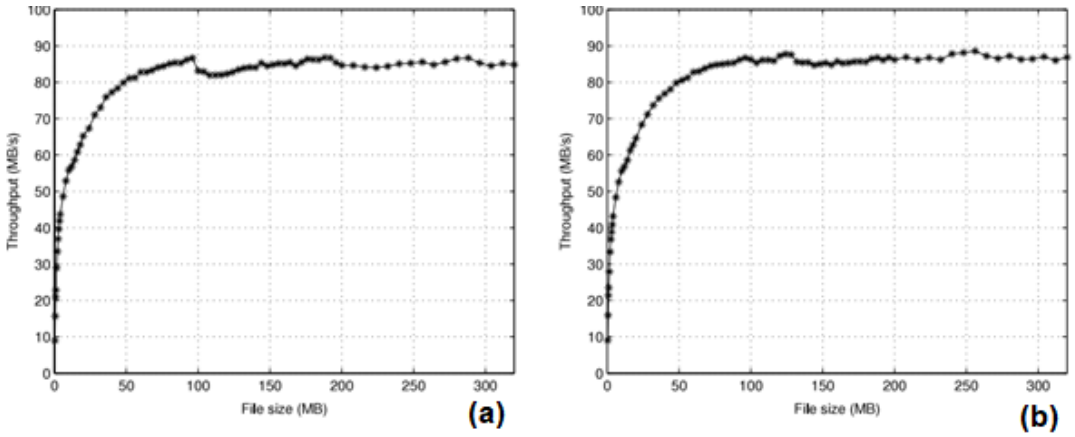
الشكل (5–2) أداء محاكاة عملية الكتابة على HDFS من أجلBS64 (b) وBS16 (a)



الشكل (5–3) أداء عملية الكتابة على HDFS من أجل BS64 (b),BS16 (a) [44]



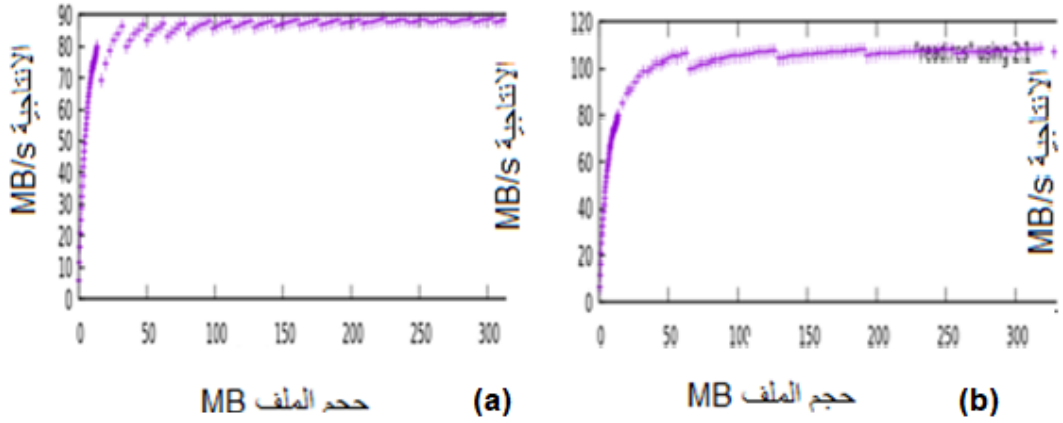
الشكل (5–4) أداء محاكاة عملية الكتابة على HDFS من أجل BS128 (b) وBS96 (a)



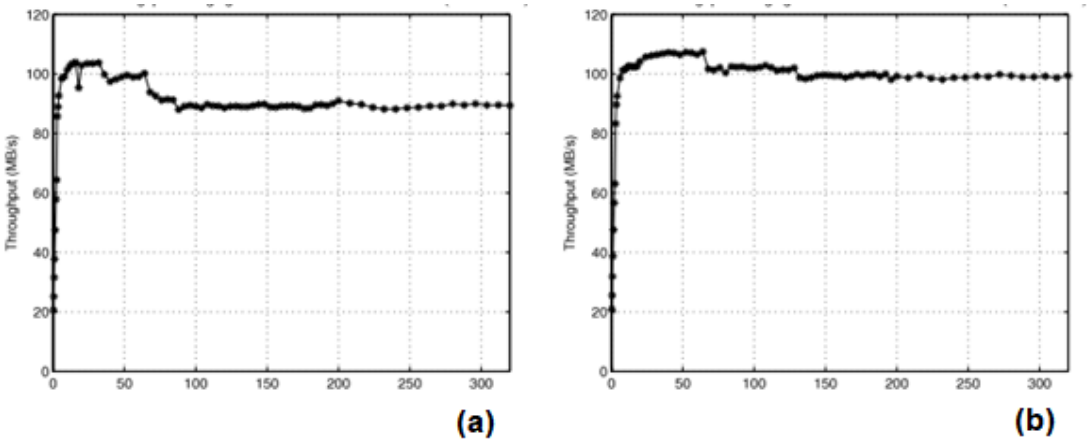
الشكل(5–5) أداء عملية الكتابة على HDFS من أجل BS128(b),BS96 (a) [44]

5–2–2 اختبار القراءة

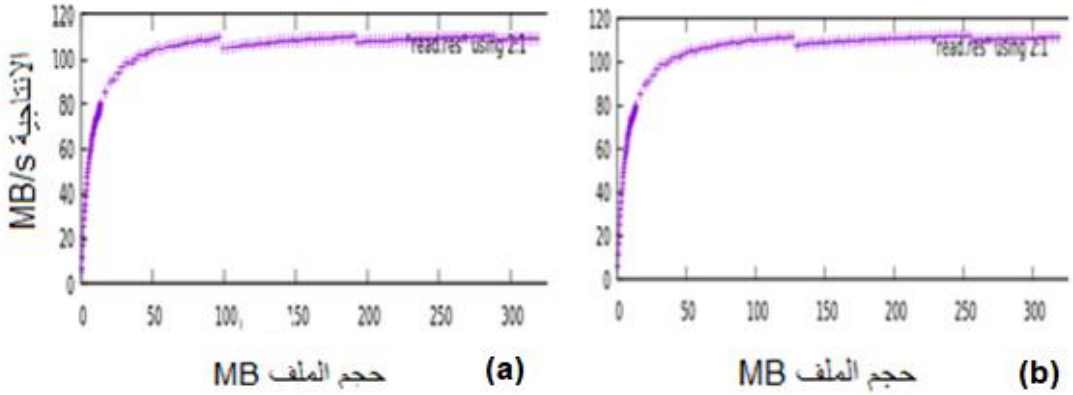
من أجل اختبار أداء عملية القراءة من HDFS قام الباحثون في [44] ومن أجل كل مجموعة بيانات بقراءة 200 ملف عشوائيا واختيار القيمة المتوسطة لزمن القراءة وتم تكرار التجربة من أجل حجوم مختلفة لـ Chunk (BS16,BS64,BS96,BS128). قمنا في هذا البحث بإجراء محاكاة لهذه التجارب بنفس الإعدادات وفيما يلي النتائج الواقعية ونتائج المحاكاة لعملية القراءة.



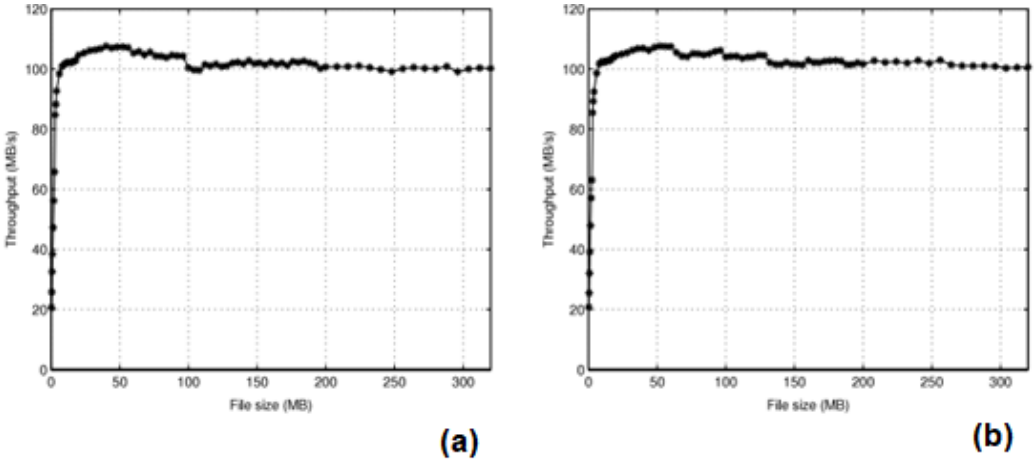
الشكل (5–6) أداء محاكاة عملية القراءة من HDFS من أجلBS64(b) و BS16 (a)



الشكل (5–7) أداء عملية القراءة من HDFS من أجل BS64(b),BS16 (a) [44]



الشكل (5–8) أداء محاكاة عملية القراءة من HDFS من أجل BS128(b) و BS96(a)



الشكل (5–9) أداء عملية القراءة من HDFS من أجل BS96,(b)BS128 (a) [44]

نتائج اختبارات HDFS ومناقشتها:

بملاحظة الخطوط البيانية للتجارب الواقعية ومقابلاتها من نتائج المحاكاة يتبين لنا أن أداء المحاكي HSG في عملية القراءة والكتابة على HDFS قريب جدا من الأداء الواقعي. ويعود سبب ذلك إلى الاعتماد على مكتبة المحاكاة SimGrid التي تعطي دقة جيدة في محاكاة الوصلات الشبكية. وكما أن النموذج المقترح للقرص الصلب يأخذ بعين الاعتبار زمن الوصول للبيانات. كما أنه ومن أجل عمليات الكتابة قمنا بمراعات تسلسل عملية الكتابة على العقد فمثلا ومن أجل معامل تكرار 3 تتم الكتابة على أول عقدة والتي تقوم بإرسال البيانات إلى العقدة الثانية ثم إلى العقدة الثالثة ويتم الإعلام عن الانتهاء من عملية الكتابة بشكل معاكس.

إن تغيير حجم الجزء في نظام الملفات الموزعة زاد من سرعة القراءة و الكتابة و لكن هذه الزيادة اصبحت طفيفة من أجل حجم جزء MB96 و 128MB.

إن وجود التعرج في المنحنيات للتجارب الواقعية وكذلك لتجارب المحاكاة يعود إلى كلفة كتابة أو قراءة جزء مقارنة بحجمه فمثلا في حال كان حجم الجزء MB 64 وكان حجم البيانات المراد كتابها 65 MB بالتالي ستتم كتابة جزئن وعملية كتابة الجزء تشتمل على تأخير ثابت يمثل زمن ارسال الأوامر وهذا الزمن ثابت بغض النظر عن حجم الجزء بالتالي ستنخفض انتاجية الكتابة أو القراءة في حالتنا لأن الجزء الأول حجمه MB 64 أما الجزء الثاني حجمه MB 1.

5–3 اختبار المحاكي

5–3–1 التحقق من دقة خرج المحاكي

من أجل التحقق من نتائج المحاكي قمنا بإعداد عنقود حاسوبي مؤلف من ثلاث عقد، إحدى هذه العقد تعمل كعقدة أسماء وكمدير موارد وفق المواصفات الموضحة بالجدول (5–2).

المعالج	Intel core 2 due 2GHz
عدد النوى	2
عدد المسالك	4
الذاكرة	4GB DDRII
القرص الصلب	Toshiba 500GB sata 2
كرت الشبكة	100 mbps
نظام التشغيل	Ubuntu 18.2

الجدول (5–2) مواصفات العقدة التي تعمل كمدير موارد وعقدة أسماء

أما العقدتين الباقيتين فكل منهما يعملان كعقدة بيانات ومدير عقدة وفق المواصفات الموضحة بالجدول (5–3)

المعالج	Intel core 2 due 2GHz
عدد النوى	2
عدد المسالك	4
الذاكرة	2GB DDRII
القرص الصلب	Toshiba 500GB sata 2
بطاقة الشبكة	100 mbps

نظام التشغيل	Ubuntu 18.2

الجدول (5–3) مواصفات العقد التي تعمل كعقد بيانات ومدراء عقد

ربطنا هذه العقد بمحول شبكي من النوع micronet وباستخدام أسلاك من نوع Stp cat6، نصبنا نظام Hadoop 3.1 على هذه العقد من أجل أجراء التجارب.

وولدنا 6 ملفات نصية بأحجام مختلفة تبدأ ب 50 Mbyte وصولا إلى 1000 Mbyte وكل ملف مؤلف من عدد من الأسطر بحيث كل سطر يحتوي على 100 Byte موزعة على عشر كلمات وكل كلمة مؤلفة من تسعة محارف ويفصل بين هذه الكلمات فراغات.

وتم نسخ هذه الملفات على نظام الملفات الموزع HDFS ومن ثم قمنا بإعداد العقدة الرئيسة والعقد الثانوية وفق الإعدادات الموضحة بالجدول (5–4).

القيمة	اسم الاعداد	الشرح
Yarn	Map–Reduce .framework.name	تحديد اسلوب تنفيذ المقابلة و الاختزال yarn/local
350	yarn.app.Map–Reduce .am.resource.mb	
256	Map–Reduce .map.memory.mb	حجم الذاكرة المخصصة للمقابل
256	Map–Reduce .reduce.memory.mb	حجم الذاكرة المخصص للمختزل
25	Map–Reduce .task.io.sort.mb	حجم العازل الذاكري
4	Map–Reduce .task.io.sort.factor	معامل الدمج
3	Map–Reduce .reduce.shuffle.parallelcopies	عدد النواسخ لجلب خرج المقابلات
0.66	Map–Reduce .reduce.shuffle.merge.percent	مقدار الذاكرة التي تبدأ بعدها عملية الدمج الذاكري للسجلات من جهة المختزل

النسبة العظمة من العازل الذاكر و التي يمكن استخدامها للدمج من جهة المختزل	mapreduce.reduce.shuffle.input.buffer.percent	0.7

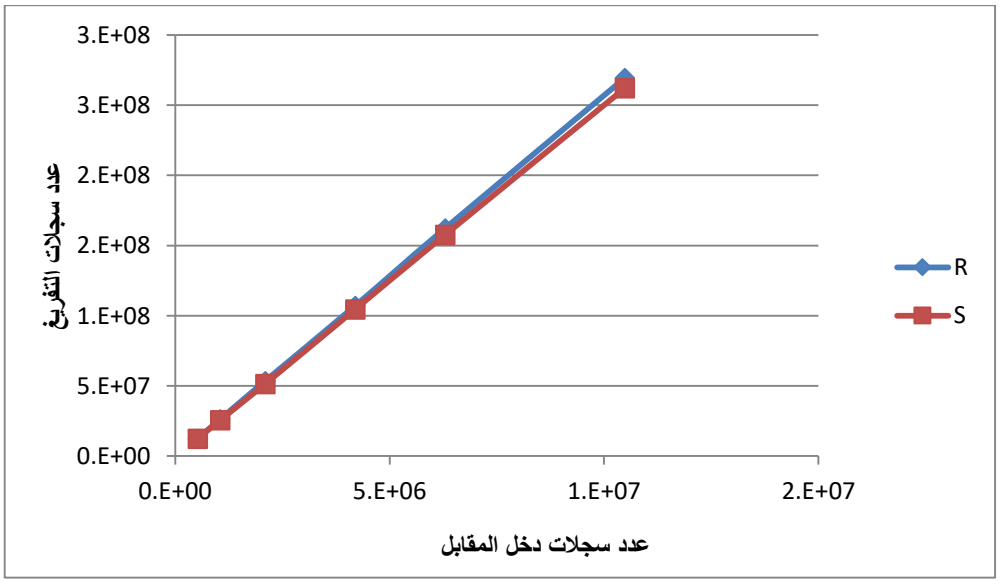
الجدول (5–4) إعدادات العقد في التجربة الواقعية

كما قمنا بتفعيل History server لتسهيل عملية تحصيل النتائج. بعد ذلك قمنا بتشغيل برنامج عداد الكلمات على هذه الملفات وقمنا بجمع القياسات الضرورية.

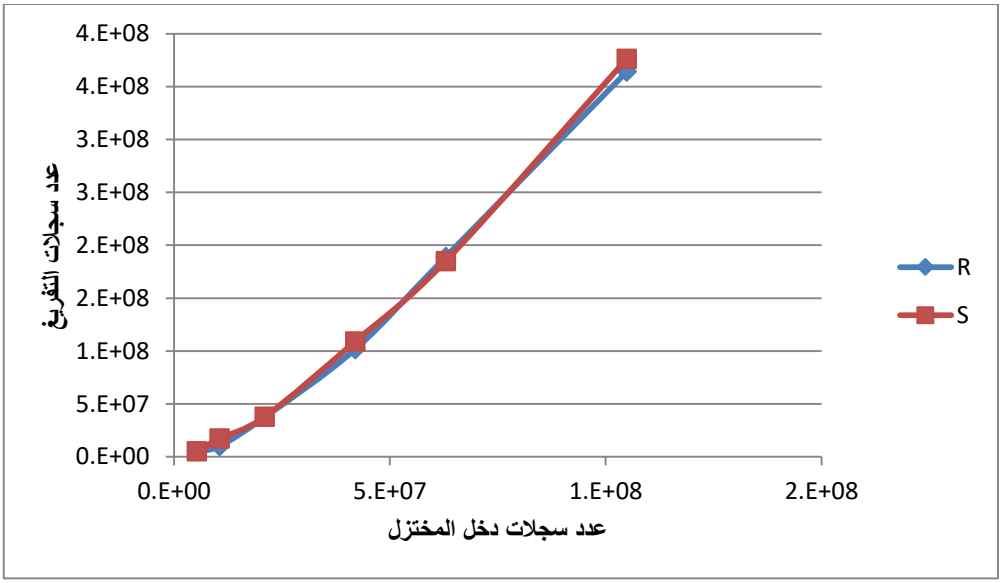
و من ثم قمنا بإعداد المحاكي وذلك عن طريق الملفات job.json,cluster.jsons .

ملاحظة: في الاختبارات سنعتبر الحرف R يمثل منحنيات التجارب الواقعية والحرف S يمثل منحنيات التجارب على المحاك.

5–3–1–1 اختبار عدد سجلات التفريغ

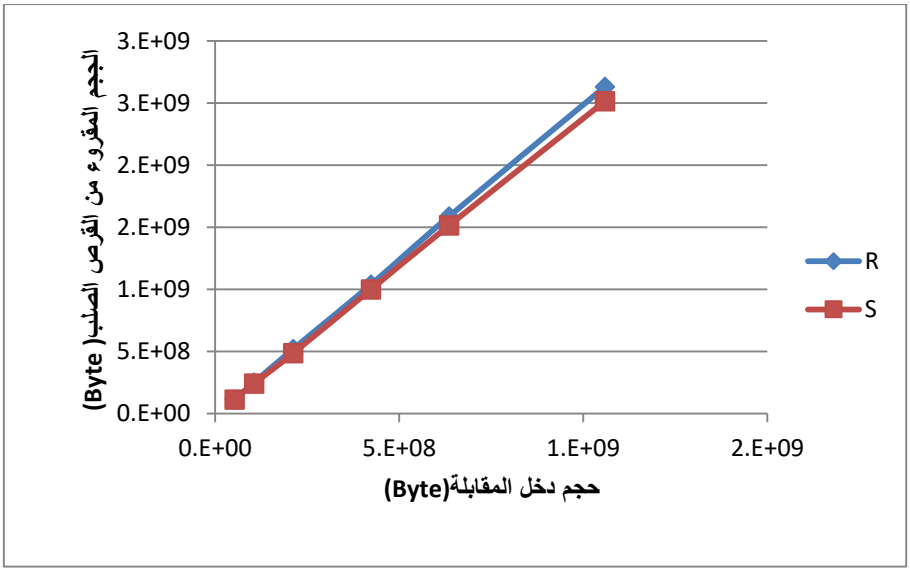


الشكل(5–10) عدد سجلات التفريغ خلال مرحلة المقابلة

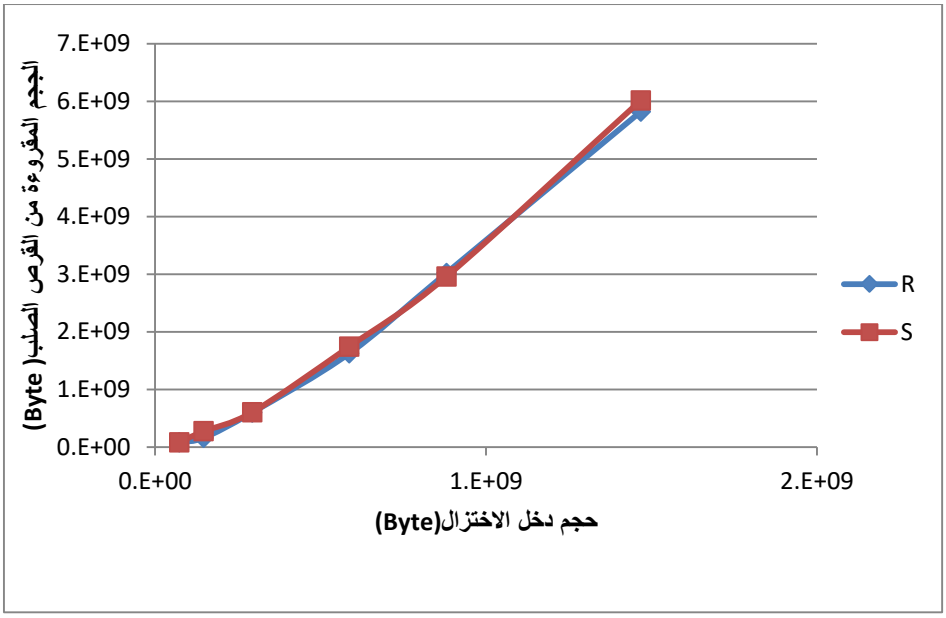


الشكل (5–11) عدد سجلات التفريغ خلال مرحلة الاختزال

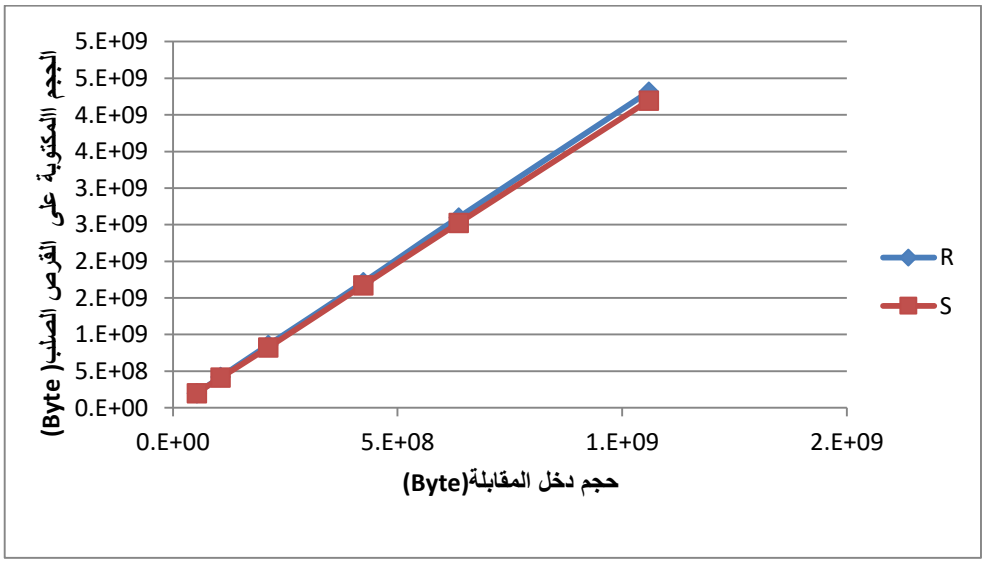
وبما أن القرص الصلب يعتبر عاملا مهما في أداء Hadoop بالتالي يجب اختبار المحاكي ومقارنة حجم البيانات المكتوبة والمقروءة من القرص الصلب في مرحلة المقابلة وفي مرحلة الاختزال.



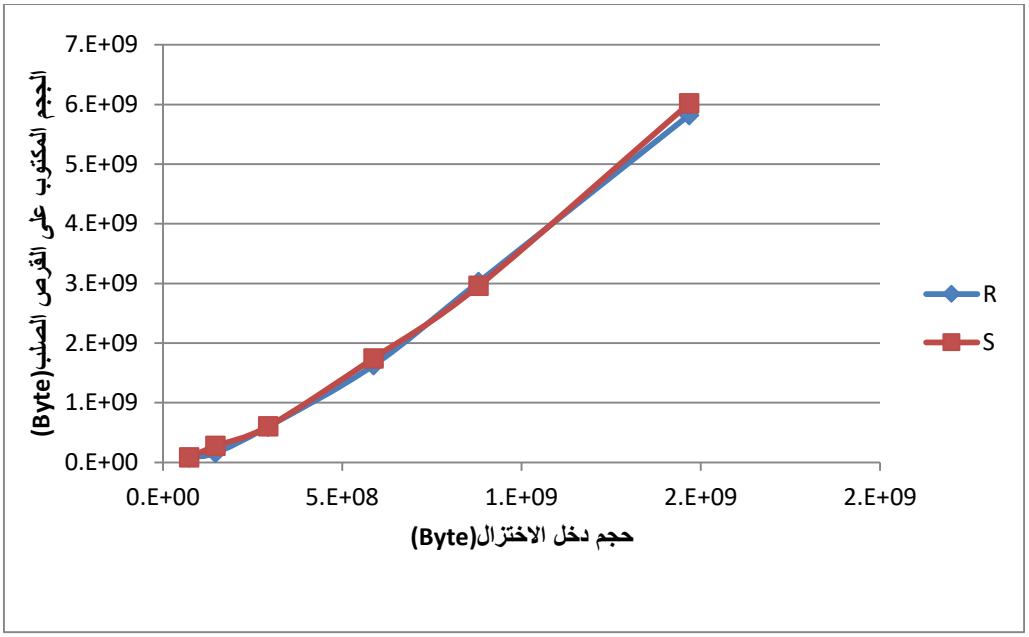
الشكل(5–12) عدد البايتات المقروءة خلال مرحلة المقابلة



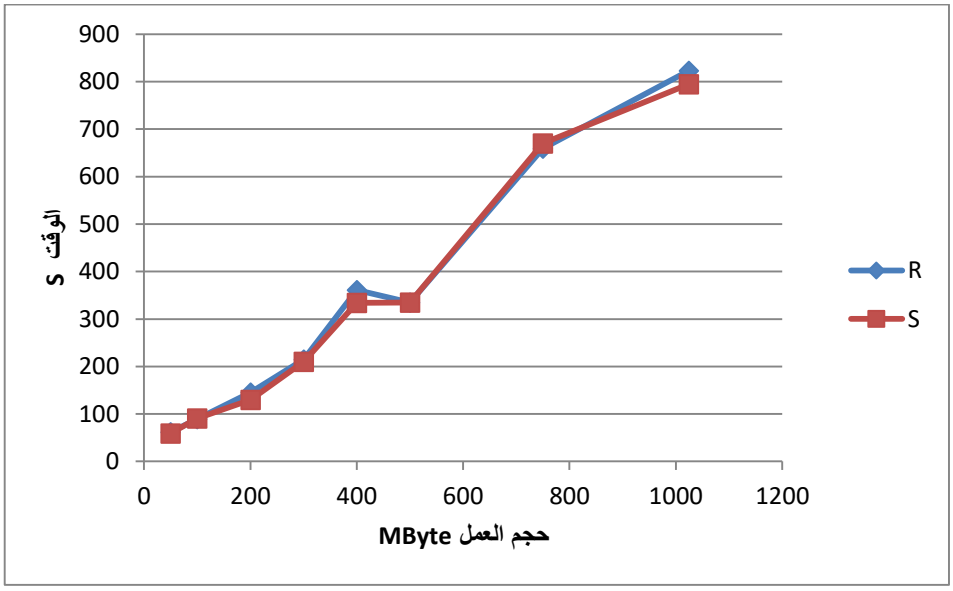
الشكل (5–13) عدد البايتات المقروءة خلال مرحلة الاختزال



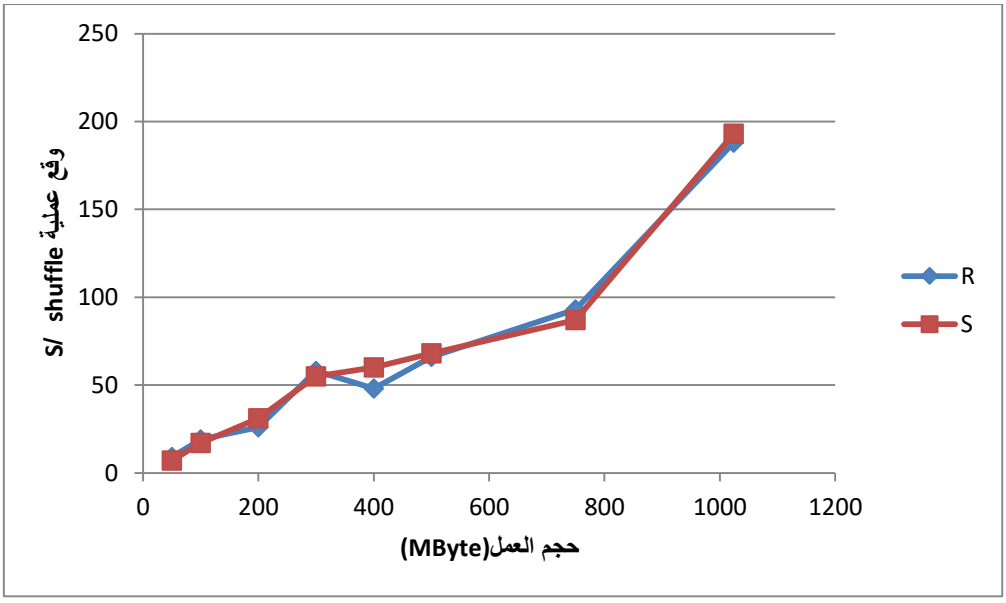
الشكل (5–14) عدد البايتات المكتوبة خلال مرحلة المقابلة



الشكل (5–15) عدد البايتات المكتوبة خلال مرحلة الاختزال



الشكل(5–16) زمن إنهاء العمل



الشكل (5–17) الزمن اللازم من أجل عملية ال shuffle

5–3–1–2 مناقشة نتائج الاختبار

1. يوضح الشكل(5–10) من أجل تجريب عدد سجلات التفريغ في المقابل وعند ضبط المتحولات الضرورية (طول سجل الدخل للمقابل وطول سجل الخرج للمقابل وحجم العازل الذاكري) كانت نسبة الخطأ لا تتجاوز 2.6%.
2. الشكل (5–11) نسبة الخطأ لا تتجاوز 7 % بالنسبة لعدد سجلات التفريغ في المختزل.
3. الشكل(5–12) نسبة الخطأ في عدد البايتات المقروءة خلال مرحلة المقابلة لا تتجاوز 4.4%
4. الشكل (5–13) نسبة الخطأ في عدد البايتات المقروءة خلال مرحلة الاختزال لا تتجاوز 3.3%
5. الشكل (5–14) كانت نسبة الخطأ في عدد البايتات المكتوبة خلال مرحلة المقابلة لا تتجاوز 2.8%.
6. الشكل (5–15) كما كانت نسبة الخطأ في عدد البايتات المكتوبة في مرحلة الاختزال لا تتجاوز 3.3%
7. إن وجود نسبة من الخطأ في عدد البايتات المقروءة والمكتوبة هو أمر طبيعية نظرا لأن هذه العملية تعتمد على عدد كبير من النشاطات العشوائية والتي لا يمكن ضبطها بشكل دقيق. وهذه النشاطات هي السرعة في توليد الخرج (انشغال المعالج) وحجمه والسرعة في الكتابة على القرص الصلب (انشغال القرص الصلب) حاولنا عند تصميم المحاكي أن نلحظ هذه

النشاطات من حيث تأثيرها على النتائج ولكن لا يمكن الوصول إلى دقة تامة نظرا لأن المحاكي يعتمد على متحولات عشوائية. كما أن الوصول إلى دقة أكبر سيدفعنا إلى تحقيق تفاصيل إضافية وبالتالي سيبطأ عملية استصدار النتائج وسيكون اعداد التجربة أكثر صعوبة.

8. إن اداء القرص الصلب كان عامل هاما في زمن الانتهاء من العمل.
9. كان هنالك صعوبة في ضبط زمن الانتهاء وذلك بسبب تغير الزمن اللازم لبدأ العمل والزمن اللازم لإقلاع المهام (في التجربة الواقعية) ويظن أن هذا الاختلاف يتعلق بتواضع المواصفات الفنية للأجهزة المستخدمة. وبشكل عام كانت نسبة الخطأ لزمن الانتهاء لا تتجاوز 8.6%

5-3-2 متطلبات الذاكرة وأزمنة التنفيذ للمحاكي

في الاختبارات التالية لدينا

- زمن اجراء التجربة يرمز إلى الزمن الذي يستغرقه المحاكي للقيام بعملية المحاكاة (الزمن الحقيقي)
- زمن المحاكاة هو زمن عملية المقابلة والاختزال والذي يقوم المحاكي باستنتاجه.
- الاستهلاك الذاكري و هو مقدار الذاكرة التي يشغلها المحاكي عند اجراء عملية المحاكاة

من أجل قياس متطلبات الذاكرة والزمن للمحاكي قمنا بالاستعانة بجهاز حاسب محمول يملك المواصفات الموضحة بالجدول (5-5)

المعالج	Intel corei5 4300M 2.6GHz
عدد النوى	2
عدد المسالك	4
الذاكرة	12GB DDRIII
القرص الصلب	500GB sata 2
نظام التشغيل	Ubuntu 20.10

الجدول (5-5) المواصفات الفنية للحاسب المستخدم في قياس الاستهلاك الذاكري والزمني للمحاك.

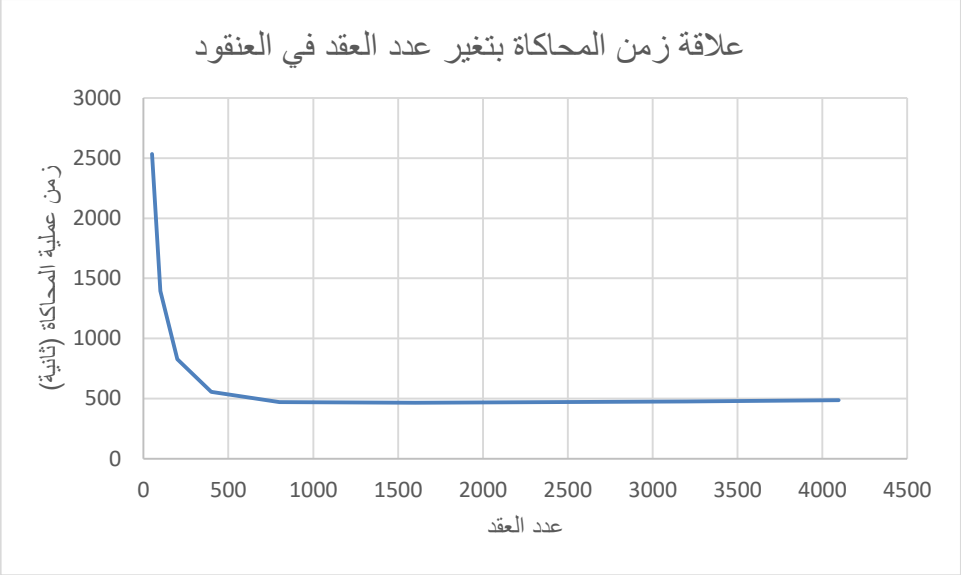
الاختبار (1): ثبات حجم البيانات مع تغير عدد العقد

إعداد الاختبار: افترضنا ان الاختبار هدفه ترتيب عدد من السجلات بالتالي عدد السجلات الداخلة يساوي عدد السجلات الخارجة، وافترضنا أن حجم البيانات هو 1TB وعدد العقد التي تقوم بالترتيب يزداد من 50 عقدة إلى 4100 والشكل(5–18) يبين اعدادات العنقود في حال 50 عقدة.

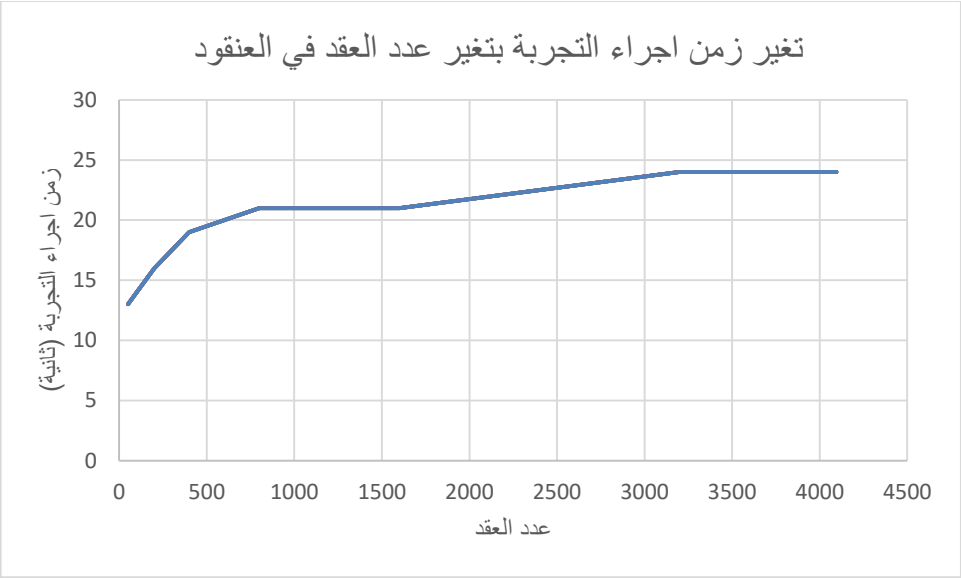
```
{   "generatePlatformAndDeploy":true,

    "racksNum":1,
    "hostsPerRack":50,
    "cpuSpeed":"3Gf",
    "coresNum":"2",
    "hddNums":3,
    "hddWriteSpeed":"120MBps",
    "hddReadSpeed":"120MBps",
    "hddreadAccess":0.014,
    "hddwriteAccess":0.014,
    "hddSlice":0.087,
    "hddCpuUseage":10,
    "accessLinkSpeed":"100MBps",
    "accessLinkLatency":"100us",
    "backBoneLinkSpeed":"1000MBps",
    "backBoneLinkLatency":"100us",
    "coreLinkSpeed":"10000MBps",
    "coreLinkLatency":"100us",


    "chunkSize":128,
    "replicatinNum":2,
    "slowStartNumFinishedMappers":5,
    "numCorePerContainer":1

}
```

الشكل (5–18) إعدادات العنقود من أجل قياس الاستهلاك الذاكري والزمني في حالة تغير عدد العقد

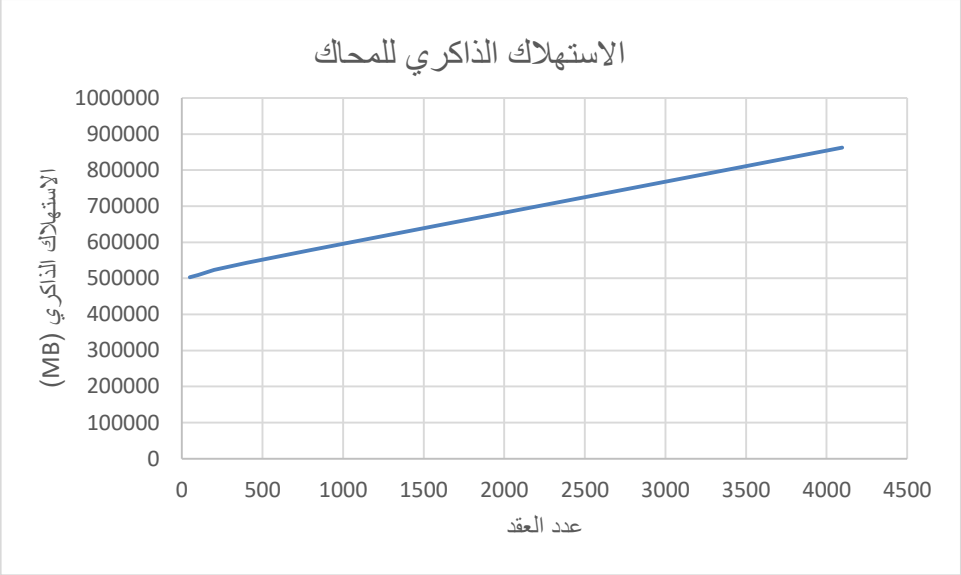
وخلص الاختبار إلى النتائج المبينة في الاشكال (5–19) و (5–20) و (5–21):



الشكل (5–19) زمن عملية المحاكاة عند تغير عدد العقد من 50–4100



الشكل(5–20) تأثير تغير العقد على نتائج عملية المحاكاة (زمن اجراء التجربة)



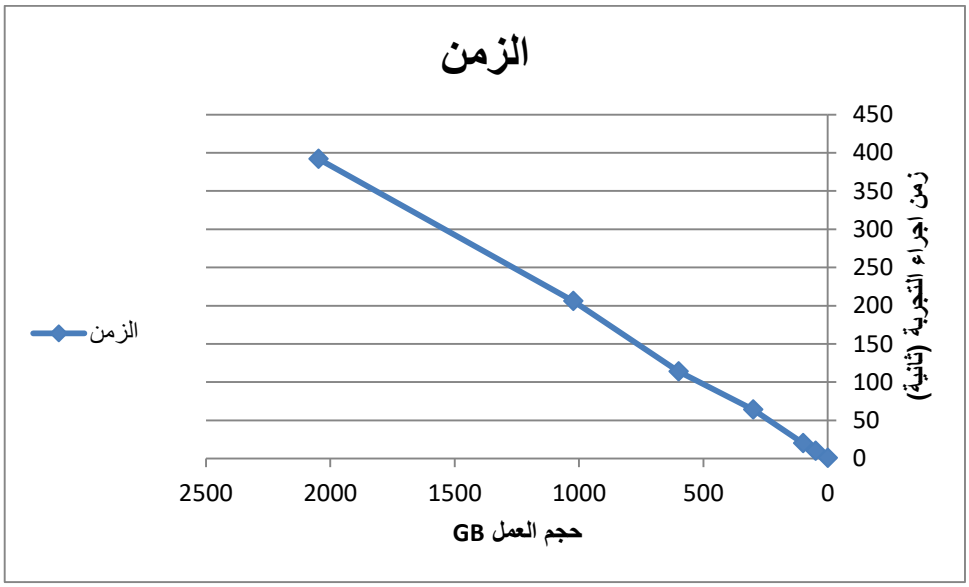
الشكل(5–21) الذاكرة التي يستهلكها المحاكي عند تغير عدد العقد من 50–4100

الاختبار (2) ثبات عدد العقد وتغير حجم العمل

قمنا بإعداد عنقود وفق الاعدادات في الشكل (5–18) وقمنا بزيادة حجم العمل من 1G إلى GB2048

و خلصنا إلى النتائج الموضحة بالأشكال (5–22) و (5–23)

الشكل(5–22) الذاكرة التي يستهلكها المحاكي عند تغير حجم العمل من 1G إلى 2048G



الشكل(5–23) زمن إجراء التجربة عند تغير حجم العمل من 1G إلى 2048G

مناقشة نتائج الاختبارين

1– من الاختبار الأول في الشكل (5–20) إن زمن اجراء التجربة يزداد بشكل طفيف، ويبدأ بالثبات تقريبا بعد 500 عقدة. لأنه من أجل كل تجربة هنالك زمن ثابت هو زمن كتابة الملفات على الـ HDFS.

2– الاستهلاك الذاكري للمحاك يزداد بشكل خطي مع زيادة عدد العقد. فعند ازدياد عدد العقد من 50 الى 4100 يزداد الاستهلاك الذاكري بنسبة 43 %.

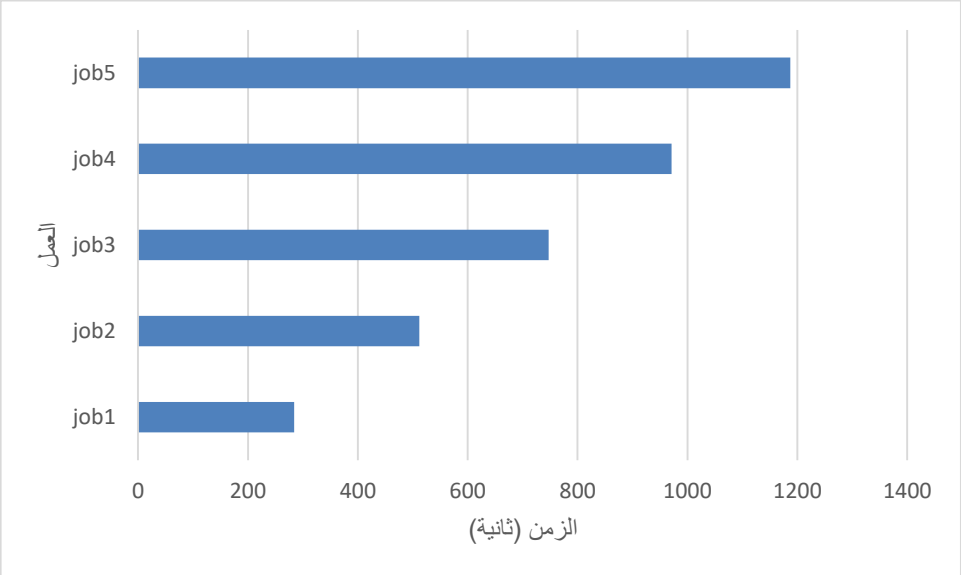
3– الشكل (5–19) ينقص زمن تنفيذ العمل مع زيادة عدد العقد ويستقر بعد عدد معين وفي حالتنا (700) حيث يبقى زمن المحاكاة ثابتا على الرغم من زيادة عدد العقد في العنقود، وذلك بسبب مرحلة ال shuffle حيث يبدأ المختزل بقراءة خرج المقابلات وبالتالي تكون عملية الترتيب والكتابة والنسخ هي عنق الزجاجة ولا يوجد أي ربح في زيادة عدد العقد بعد هذا الحد.

4– من الشكل (5–22) والشكل (5–23) نلاحظ ان زمن اجراء التجربة والاستهلاك الذاكري للمحاكي يتناسب بشكل طردي مع زيادة حجم البيانات، ونلاحظ ان المحاكي مستهلك للذاكرة

بشكل كبير وذلك لأنه يمثل معظم تفاصيل عملية المقابلة والاختزال فمثلا ومن اجل عمل حجمه 1TB وحجم جزء 128MB سيتم حجز 8192 غرض يمثل كل منها جزء من البيانات وفي حال معامل التكرار كان 2 سيتم حجز ضعف هذا العدد وسيكون لدينا 8192 عملية مقابلة بالإضافة للتفاعلات خلال عملية التفريغ على القرص الصلب ومعالجة البيانات. إن هذا التعقيد الذاكري والزمني يعتبر عادية مقارنة بعدد التفاعلات التي لحظناها والتي كانت الغاية منها القدرة على قياس تأثير التنافسية على الموارد (الشبكة القرص، المعالج، القرص الصلب) وذلك عند تنفيذ أكثر من عمل وهذا ما سنبينه في التجربة التالية.

5-3-3 تأثير أنماط الجدولة على زمن انهاء الاعمال

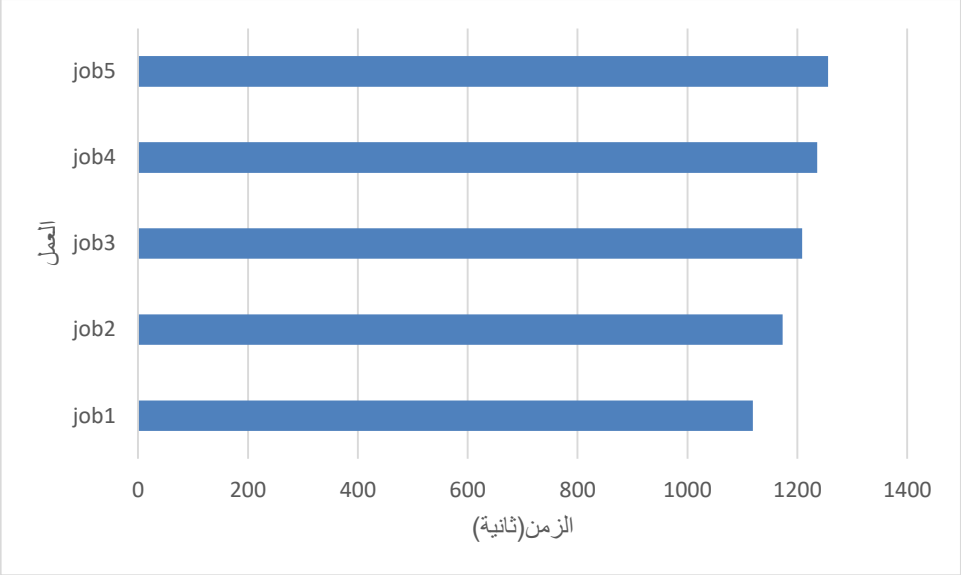
حققنا جميع أنواع الجدولة التي يدعمها Hadoopوهي (Fair,FIFO,capacity) ولتبيان تأثير نمط الجدولة على زمن انجاز العمل قمنا بإرسال 5 اعمال ترتيب متماثلة الى العنقود وهذه الأعمال بحجم GB100 ثم غيرنا نوع الجدولة وحصلنا على النتائج الموضحة بالأشكال (5-24) و (5-25) و (5-26) و (5-27)



الشكل(5-24) زمن المحاكاة في حالة نمط الجدولة (FIFO)

معدل تأخر تخديم العمل في الرتل	17ثانية
معدل طول الرتل	3.98
استخدامية العنقود	0.121

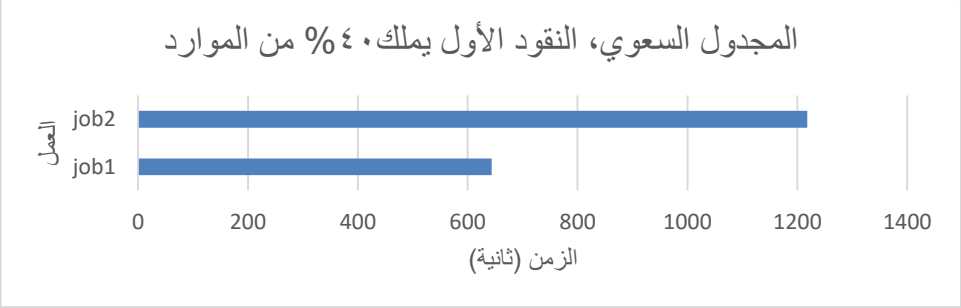
الجدول (5-6) احصائيات الرتل (FIFO)

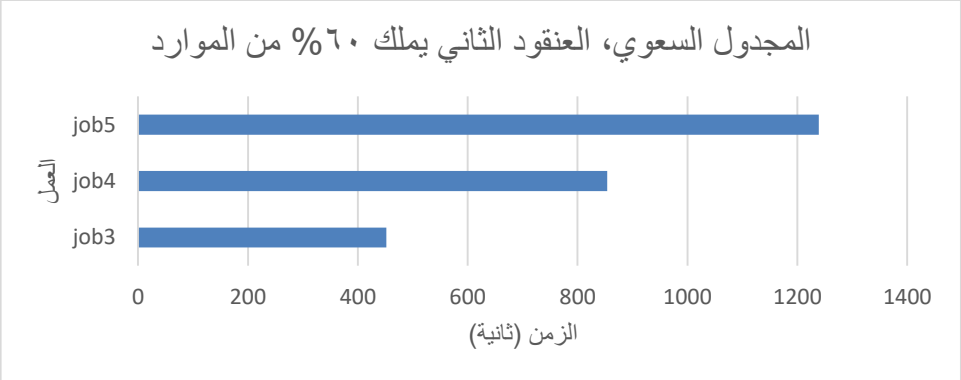


الشكل(5–25) زمن المحاكاة في حالة نمط الجدولة (FAIR)

معدل تأخر تخديم العمل في الرتل	5.9 ثانية
معدل طول الرتل	4.15
استخدامية العنقود	0.138

الجدول (5–7) احصائيات الرتل (fair)





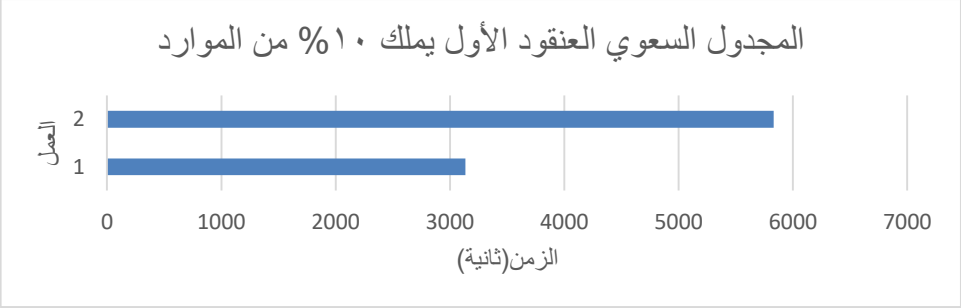
الشكل(5–26) زمن المحاكاة في حالة نمط الجدولة (Capacity)

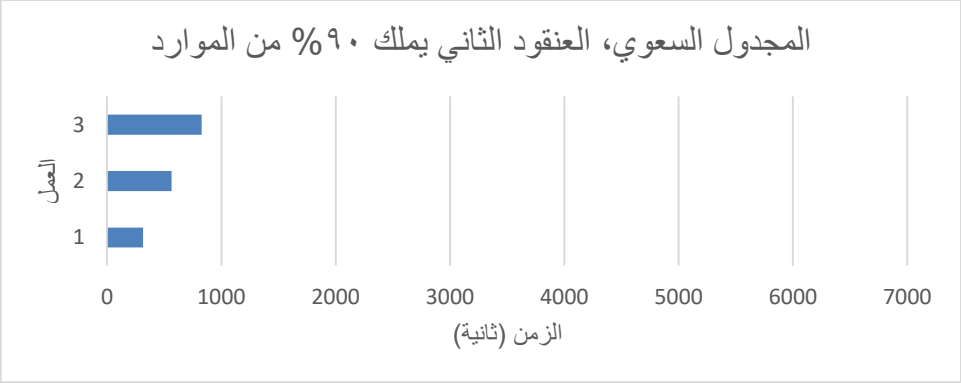
معدل تأخر تخديم العمل في الرتل	17.9ثانية
معدل طول الرتل	1.6
استخدامية العنقود	0.138

الجدول (5–8) احصائيات الرتل (capacity q1 40%)

معدل تأخر تخديم العمل في الرتل	9. ثانية
معدل طول الرتل	2.2
استخدامية العنقود	0.104

الجدول (5–9) احصائيات الرتل (capacity q2 60%)





الشكل(5–27) زمن المحاكاة في حالة نمط الجدولة (Capacity)

معدل تأخر تخديم العمل في الرتل	18 ثانية
معدل طول الرتل	1.5
استخدامية العنقود	0.4

الجدول (5–10) احصائيات الرتل (q1 capacity10%)

معدل تأخر تخديم العمل في الرتل	18 ثانية
معدل طول الرتل	2.03
استخدامية العنقود	0.06

الجدول (5–11) احصائيات الرتل (q2 capacity90 %)

إن هذا الاختبار يساعد في تحديد نمط المحاكاة المناسب لتجربة معينة. هذا ويمكن دعم المحاك بأنماط جدولة أخرى وذلك بإضافة صف يرث من الصف YarnSchedulerBace بحيث يحقق هذا الصف الدوال (freeContainer,allocate,addreq,addjob) هذه الإمكانية تمكن الباحثين من تطوير خوارزميات جدولة مخصصة وتجربة أدائها.

5–3–4 مقارنة HSG مع باقي المحاكيات

في الجدول(5–11) التالي مقارنة بين عدد من محاكيات المقابلة والاختزال والمحاكي HSG من حيث الميزات واسلوب التحقيق.

الاسم	دعم الجدولة	عقد مفصة	المكتبات المستخدمة والتحقيق	محاكاة الشبكة	محاكاة عدة أعمال في نفس الوقت	تحمل الفشل	إصدار Hadoop	نوع المحاكاة
Mrperf [12]	FIFO	نعم	,tcl,c++, Python	NS2	نعم	نعم	1	DES
MRSim[18]	FIFO,fair	نعم	Simjava ,java	GridSim	نعم	لا	1	DES
MRtune[19]	لا يدعم	لا	غير معروف	غير معروف	لا	لا	1	غير معروف
HSim[21]	FIFO,fair	نعم	Java	لا يوجد	نعم	لا	1	DES
MRSG[26]	FIFO	لا	C	SimGrid	نعم	لا	1	DES
YARNSim [23]	FIFO	نعم ،لكن هنالك محدودية بحساب تأخير المعالجة	C ,ROSS	CODES	نعم	نعم	2	PDES
WaxElephant[28]	FIFO وFair وCapacity	نعم	Java	لا يوجد	نعم	لا	1	DES
HSG	FIFO وFair وCapacity	نعم	C++	SimGrid	نعم	نعم	2	DES

الجدول (5–12) مقارنة بين عدد من محاكيات المقابلة والاختزال مع المحاكي HSG (Hadoop SimGrid)

و من أجل مقارنة زمن تنفيذ المحاكي HSG مع باقي المحاكيات استفدنا من التجارب التي قام بها Wagner Kolberg و آخرون على المحاكي MRSG [26] حيث أنجزوا عمل ترتيب على المحاكيات MRSG و MRSim و MRperf و اعدادات العمل و العنقود موضحة بالجدول (5–12)

	Config. 1
Nodes	32
Cores	64
CPU	AMD Opteron 246 2.0 GHz
Memory	2 GB
Bandwidth	1 Gb/s
Input	12 GB
Maps	192
Reduces	64
Map output	12 GB

الجدول (5–13) اعدادات العمل و العنقود المستخدمة في [26]

وخلصوا إلى النتائج الموضحة بالجدول (5–14)

MRSim	MRPerf	MRSG
+12 h	3m 37.724 s	0.401s

الجدول (5–14) زمن انجاز التجربة بالنسبة لـ MRSG و MRPerf و MRSim [26]

كررنا هذه التجربة من أجل الاعدادات نفسها الموضحة بالجدول(5–13) حيث شغلنا MRSG و HSGعلى جهاز حاسب يمتلك المواصفات الموضحة بالجدول(5–5) و من اجل الاعدادات الموضحة بالجدول (5–12) و خلصنا للنتائج الموضحة بالجدول (5–15)

HSG	MRSG
2.1s	0.490

الجدول (5–15) زمن انجاز التجربة بالنسبة لـ MRSG وHSG

على الرغم من أن النتائج في الجدول (5–15) تمت على عتاد حاسوبي مختلف عن التجربة التي قام بها الباحثون في الجدول (5–14) إلى أن زمن انجاز التجربة بالنسبة للمحاكي MRSG متقارب في

كلا التجربتين. يمكننا أن نستنبط من مقارنة الجدولين أن هنالك كسب في الوقت بشكل كبير عند استخدام HSG مقارنة بـ MRSim و MRPerf فمثلا HSG اختزل زمن التجربة بمقدار 99% و لكن MRSG يبقى اسرع من البقية، و يعود ذلك للأسباب الآتية:

1- MRperf يراعي تفاصيل العقد و لكنه يعتمد على المحاكي NS2 وهو محاك على مستوى الحزمة
2- MRSim يراعي تفاصيل العقد و لكنه يعتمد على المكتبة SimJava والتي تسبب بطء المحاكي بسبب اعتمادها على المسالك البرمجية مباشرة كما أنها محققة باستخدام لغة Java.
3- HSG إن هذا المحاكي أعطى سرعة جيدة على الرغم من تحقيقه تفاصيل كثيرة جيدة و ذلك لأنه يعتمد على الواجهة S4U من المكتبة SimGrid و التي تتبع نفس النموذج البرمجي Actor model المستخدم هنا.
4- MRSG يتعامل مع المهام ككتلة واحدة مما يبسط عمية الحساب و بالتالي اعطى أعلى سرعة و لكنه يغفل عن الكثير من التفاصيل و التنافسية بين الأعمال (المراعات بباقية المحاكيات)، هذا التبسيط يقوم على حساب الدقة.

5-4 الخاتمة

عرضنا في هذا الفصل التجارب التي تمت على المحاكي HSG وهذا التجارب كانت على أنواع عدة.

1- اختبارات لضبط الأداء (التحقق من دقة نظام الملفات الموزع والتحقق من دقة نتائج المحاكي مقارنة بالتجارب الواقعية)

2- اختبارات قياس الموارد (قياس متطلبات الذاكرة والاستهلاك الزمني للمحاكي)

3- اختبارات توضح بعض الميزات (دعم أنماط الجدولة المختلفة)

4- اختبارات تقارن أداء المحاكي مع باقي المحاكيات (مقارنة أداء المحاكي مع المحاكي MRSG)

قمنا بعد كل اختبار بمناقشة المنحنيات والنتائج مبينين مكامن القوة والضعف في المحاكي HSG.

الفصل السادس

الخاتمة والآفاق المستقبلية

6–1 المقدمة

يلخص هذا الفصل العمل الذي تم إنجازه ضمن هذا البحث وهو تطوير محاك لإطار المقابلة والاختزال، كما سنستعرض في هذا الفصل المساهمة التي تقدمها هذه الاطروحة مع تلخيص للنتائج ومن ثم المقترحات.

6–2 ملخص الأطروحة

إن إطار المقابلة والاختزال تم اقتراحه من قبلGoogle [1] في عام 2004 من أجل معالجة البيانات الضخمة، وظهر فيما بعد العديد من التحقيقات لهذا الإطار، إلا أن أهمها وأكثرها انتشارا هو Hadoop(Hadoop).

إن عملية معالجة البيانات الضخمة تختلف باختلاف المهمة المنفذة والبيانات المراد معالجتها. فالمهام تختلف عن بعضها من حيث:

1– استهلاكها للمعالج.

2– القراءة والكتابة على القرص الصلب.

3– القراءة والكتابة على نظام الملفات الموزع.

4– القراءة والكتابة على الشبكة.

5– الاستهلاك الذاكري.

أما البيانات فتختلف عن بعضها من حيث:

1– الحجم.

2– التوزع على العقد والأقراص.

3- عدد النسخ.

4- الصيغ المخزنة.

5- الضغط والتشفير.

إن هذه الاختلافات وغيرها تؤثر في زمن تنفيذ المهام وفي استهلاك الموارد، وبالتالي أي تعديل بسيط في إعدادات المهام والعناقيد الحاسوبية يصبح له أثر واضح على الأداء وعلى استهلاك الموارد ويصبح الوصول إلى أفضل أداء أمرا صعبا إن لم يكن مستحيلا. فبعض الإعدادات تناسب حالات معينة ومهام معينة وقد لا تتناسب مع حالات أخرى. هنا ندرك اهمية وجود محاكٍ لإطار المقابلة والاختزال يساعد في ضبط أعمال المقابلة والاختزال.

إن نموذج المقابلة والاختزال يعتبر نموذجا جديدا نسبيا كما أن MRPerf وMumak [5] هما أول محاكيان لهذا النموذج. علما أنه هنالك تشابه بين نموذج المقابلة والاختزال والحوسبة الشبكية (Gridcomputing) إلا أن محاكيات الحوسبة الشبكية لا يمكنهم محاكاة إطار المقابلة والاختزال بشكل صحيح. لأن هذه المحاكيات تقوم بنمذجة الأعمال المرسلة للنظام كأعمال دفعية (batch jobs) حيث ان كل عمل يشتمل على كلفة حسابية معينة ويحتوي على توصيفات مسبقة للدخل وللخرج. وبالتالي لا بد من محاكيات مخصصة لنموذج المقابلة والاختزال والجدول التالي يلخص اهم هذه المحاكيات وفي السطر الاخير المحاكي HSG الذي قمنا بتحقيقه في هذه الأطروحة.

الاسم	دعم الجدولة	عقد مفصة	المكتبات المستخدمة والتحقيق	محاكاة الشبكة	محاكاة عدة أعمال في نفس الوقت	تحمل الفشل	إصدار Hadoop	نوع المحاكاة
Mrperf [12]	FIFO	نعم	,tcl,c++, Python	NS2	نعم	نعم	1	DES
MRSim[18]	FIFO,fair	نعم	Simjava ,java	GridSim	نعم	لا	1	DES
MRtune[19]	لا يدعم	لا	غير معروف	غير معروف	لا	لا	1	غير معروف
HSim[21]	FIFO,fair	نعم	Java	لا يوجد	نعم	لا	1	DES
MRSG[26]	FIFO	لا	C	SimGrid	نعم	لا	1	DES
YARNSim [23]	FIFO	نعم ،لكن هنالك محدودية بحساب تأخير المعالجة	C ,ROSS	CODES	نعم	نعم	2	PDES
WaxElephant[28]	FIFO وFair وCapacity	نعم	Java	لا يوجد	نعم	لا	1	DES

الجدول (6–1) مقارنة بين محاكيات المقابلة والاختزال والمحاكي HSG.

ولقد اعتمدنا على المحاكي SimGrid في تحقيق المحاكي HSG والذي تم تصميمه بحيث حقق المتطلبات التالية:

1– قابل للتوسعة بشكل كبير بحيث نستطيع اختبار أثر التوسعية على أداء المقابلة والاختزال.

2– يدعم نسخة Map–Reduce 2.

3– المقدرة على محاكاة عقد مخصصة من حيث القرص الصلب والمعالج.

4– المقدرة على محاكاة الشبكة الحاسوبية بشكل جيد والتي تعتبر عامل هام في أداء المقابلة والاختزال.

5– تقديم سرعة جيدة في تنفيذ عملية المحاكاة.

6– تقديم آلية لتسهيل عملية إعداد وإجراء التجارب.

7– تقديم آلية للحصول على النتائج والمتحولات الإحصائية بشكل سهل وواضح.

8– المقدرة على محاكاة عدة أعمال في نفس الوقت.

9– المحاكي HSG يدعم عدة خوارزميات جدولة.

. بعدها تحققنا من نتائج هذا المحاكي وقمنا بمقارنة أدائه مع المحاكيات MRSG و MRPerf.

6–3 المساهمة التي قدمتها هذه الاطروحة

1– دعم المحتوى العربي بشرح مفصل عن إطار المقابلة والاختزال مبيناً مبدأ عمله مع أمثلة مفصلة، بحيث يمكن استخدام الفصل الثاني من هذه الأطروحة كفصل في مرجع أو محاضرة جامعية ضمن مواد النظم الموزعة أو تحليل البيانات الضخمة.

2– إعداد دراسة مرجعية تفصيلية لمحاكيات المقابلة والاختزال وقد ختمنا هذه الدراسة بجدول يلخص خصائص كل محاكي.

3– دراسة عدد من المحاكيات الشبكية والتوصية باعتماد المحاكي SimGrid من أجل نمذجة الشبكة في أطر البيانات الضخمة.

4- وضع تصميم عام للمحاكي مزود بعدد من المخططات التسلسلية والمخططات التدفقية لتوضيح آلية عمله، يصلح هذا النمط من التصميم لاعتماده لتصميم محاكيات لنظم عنقودية أخرى من نوع المقابلة والاختزال (Spark[7] مثلاً)

5- تحققنا من دقة المحاكي كما قارنا أداءه مع باقي المحاكيات.

6- تصميم المحاكي بحيث يدعم أنماط جدولة عدة وترك المجال للباحثين لتصميم خوارزميات الجدولة الخاصة بهم وتضمينها للمحاكي بسهولة.

6-4 ملخص النتائج التي تم الحصول عليها

1- ان اعتماد المحاكي على مكتبة SimGrid، ومراعاة عمليات تكرار البيانات وتسلسل عمليات القراءة والكتابة على القرص الصلب، كان له إثر جيد في الوصول إلى دقة جيدة من حيث سرعة القراءة والكتابة على نظام الملفات الموزعة وذلك بمقارنة نتائج المحاكاة مع نتائج التجربة الواقعية.

2- استخدام ملفات من نوع JSON سهل عملية إجراء التجربة كما أن تصدير خرج المحاكاة كملف HTML يسهل قراءة ومناقشة النتائج.

3- وجود عدد كبير من المعاملات ساعد في زيادة دقة التجارب لكن ذلك جعل من إعداد التجربة أكثر صعوبة، ولكن هذا الكم من المعاملات يعتبر قليلاً مقارنة بالنظام الواقعي (Hadoop) والذي يحتوي على المئات من المعاملات.

4- إن النمذجة الجيدة للمحاكي أدت إلى زيادة عدد مكونات البرنامج وبالتالي زيادة الاستهلاك الذاكري للمحاكي، لكن اعتماد بنية طبقية في التصميم ساهم في ضبط عملية تعقيد المكونات وفتح المجال لإضافة المزيد من الوظائف إلى المحاكي عن طريق الباحثين المهتمين بالمستقبل. إذا، يدعم المحاكي التطوير والإضافة على مكوناته.

5- إن نموذج العمال Actor Model المستخدم في SimGrid والذي استخدم في هذا المحاكي ساهم بزيادة التوسعية للمحاكي وزاد من سرعة المحاكي وكان مناسبا بشكل كبير نظرا لأن النموذج الذي نقوم بمحاكاته هو نموذج موزع بالأساس.

6- إن زمن انجاز التجربة كان قليلا نسبيا مقارنة بالمحاكيات التي تراعي تفاصيل العقد و تفاصيل المهام (MRSim, MRPerf) و لكن هذا الزمن هو بالتأكيد اكبر من زمن انجاز التجربة بالنسبة للمحاكيات التي لا تراعي تفاصيل العقد و لا تفاصيل المهام مثل (MRSG)

6-5 المقترحات المستقبلية

1- استطاع المحاكي محاكاة أعمال أكبر من 2TB على عنقود يحتوي 4000 عقدة، بالتالي هنالك قدرة للمحاكي على التوسع بشكل كبير ويعود ذلك إلى اعتماده على المكتبة SimGrid التي تقوم بتمثيل العقد كـActors وليس كمسالك. لذلك يوصى باستخدام هذه المكتبة لمحاكاة التطبيقات الموزعة التي تحتوي على عدد كبير من العقد.

2- يمكن تزويد المحاكي بميزة تحمل الفشل التي تحدثنا عنها في الفصل الثاني وهذا ممكن نظرا للنمذجة الجيدة للنظام.

3- يمكن تزويد المحاكي بالقدرة على حساب الطاقة المستهلكة في العنقود نظرا لأن المكتبة SimGrid تدعم ذلك، ولم نحقق هذه الميزة لعدد اسباب وأهم هذه الأسباب عدم القدرة على مقارنة نتائج المحاكاة بنتائج تجارب واقعية.

4- كتابة أنواع إضافية من المجدولات على هذا الإطار وتجربتها على المحاكي.

5- يمكن استخدام هذا المحاكي لإجراء بعض الدراسات على أطر تعتمد على المقابلة والاختزال مثل Pig [47] و Hive [48].

6- بسبب الفصل بين المكونات والنمذجة الجيدة لنظام الملفات الموزع يمكن استخدام هذا الجزء من المحاكي (نظام الملفات الموزع) في اجراء بعض الدراسات على الاطر التي تعتمد عليه مثل HBase [49].

7- عند محاكاة أي نظام يجب دراسة هذا النظام بشكل مفصل، ومن ثم تحديد الهدف من عملية المحاكاة مما يساعد في بناء النموذج المناسب. كما أن دراسة النظام بشكل جيد تساعد في اختيار الأدوات المناسبة لمحاكاة هذا النظام ولعل دراسة النظام واختيار الأدوات يستهلك الكثير من الجهد والوقت مقارنة بمرحلة التحقيق وجمع النتائج.

الفصل السابع

الملاحق

الملحق (1)

إن هذا الملحق يستند بشكل كامل على المرجع [50].

1– محاكاة الأحداث المتقطعة

المحاكاة (Simulation): هي عملية تقليد للنشاطات أو المهام الخاصة بمختلف أنواع الأدوات المستخدمة في حياتنا الواقعية، الأداة أو العملية التي نهتم بمحاكاتها تدعا نظاما (System)، وعادة ما يكون علينا وضع العديد من الفرضيات عن طبيعة عمل النظام من أجل دراسته بشكل واضح. يطلق على هذه الفرضيات اسم النموذج (model) وغالبا ما تكون الفرضيات على شكل علاقات رياضية أو منطقية. هذه العلاقات تستخدم لفهم طبيعة سلوك النظام الممثل باستخدام النموذج.

إذا كانت العلاقات التي تشكل النموذج بسيطة بشكل كافٍ فإنه من الممكن استخدام الطرق الرياضية (الجبر أو الاحتمالات ..الخ) من أجل الحصول على معلومات مطابقة لسؤال معين، وهذا ما يدعى الحل التحليلي. لكن معظم الأنظمة الواقعية معقدة كثيرا ومن غير الممكن تقييم النماذج الخاصة بها بشكل تحليلي، وهذه النماذج لابد من دراستها عن طريق المحاكاة. إن المحاكاة تستخدم الحاسوب الآلي من أجل تقييم النموذج بشكل عددي، وبعد تقييم النموذج يتم جمع البيانات من أجل تخمين الخصائص الصحيحة للنموذج.

ولعله من أهم مجالات تطبيق المحاكاة:

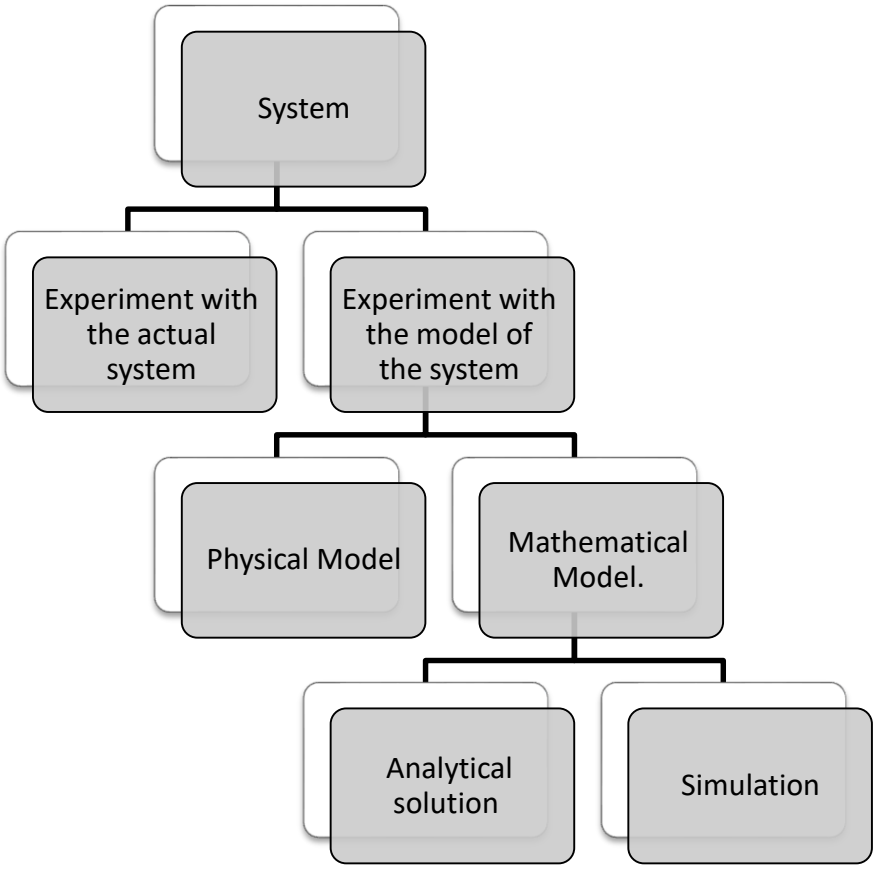
- تصميم أنظمة التصنيع وتحليلها.
- تقييم أنظمة الأسلحة الحربية أو متطلبات تموينها.
- تحديد البروتوكولات أو المتطلبات العتادية من أجل شبكات الاتصال.
- تحديد المتطلبات البرمجية والعتادية لأنظمة الحاسوب.
- تصميم ونظم النقل وتشغيلها في المطارات والطرق السريعة والبوابات والمترو.
- تقييم التصميم للمنظمات التي تقدم خدمات معينة مثل مراكز الاتصال والمستشفيات ومكاتب البريد.
- تحليل الأنظمة المالية.

2 الأنظمة والنماذج والمحاكاة (Systems ,models and simulation)

النظام: هو عبارة عن مجموعة من الكيانات (أشخاص، آليات) تتفاعل فيما بينها لإنجاز منطق معين في النهاية. ومعنى كلمة نظام من الناحية العملية يتعلق بهدف دراسة معينة. مجموع الكيانات التي تشكل النظام في دراسة معينة يمكن أن يكون مجموعة جزئية من نظام أشمل. على سبيل المثال إذا أراد أحدهم دراسة بنك لتحديد عدد أمناء الصناديق من أجل تأمين خدمة كافية للزبائن، علما أن الزبون يمكن أن يقوم بعمليتين هما السحب والإيداع، وبذلك يمكن تعريف النظام كمجموعة من أمناء الصناديق والزبائن الذين ينتظرون في الدور والزبائن الذين يحصلون على الخدمة. وإذا تم تضمين مدير البنك والحارس فلا بد من إضافتهما على النموذج ليصبح النموذج الجديد أشمل.

نعرف حالة النظام (System state) بأنها مجموعة من المتحولات الضرورية لشرح حالة النظام في لحظة زمنية معينة، وبالعودة لمثال البنك وكمثال عن متحولات الحالة التي يمكن أن نستخدمها: عدد أمناء الصناديق العاطلين عن العمل وعدد الزبائن في البنك ووقت وصول كل زبون إلى البنك

ومن الضروري للعديد من الأنظمة أن نقوم بدراستها لنتعرف أكثر على العلاقات بين مكوناتها المختلفة، أو لنتوقع الأداء تحت ظروف جديدة معتبرة. الشكل (2–1) يبين لنا الطرق المختلفة لدراسة النظم.



الشكل(1) طرق دراسة النظام

3 التجربة مع النظام الواقعي مقابل التجربة مع نموذج النظام

لعله من المقبول أن تقوم بتغيير النظام بشكل فيزيائي وجعله يعمل بعد ذلك تحت الشروط الجديدة طبعا إذا كان ذلك ممكنا أو مجديا ماديا. لكن من النادر أن تتم الدراسة بهذا الشكل لأن بعض التجارب يمكن أن تكون مكلفة جدا أو أن تسبب ضرر للنظام. على سبيل المثال، يمكن أن يقرر البنك تقليل عدد أمناء الصناديق لتقليل التكاليف، لكن يمكن أن يؤدي تجربة هذا الإجراء إلى التسبب في انتظار الزبائن لفترات طويلة حتى يتم تخديمهم. يمكن أن يكون النظام غير موجود ونريد دراسته من أجل عدة إعدادات مفروضة لنحدد كيف يتوجب علينا أن نقوم ببنائها، وكمثال عن هذه الحالة بناء شبكة اتصالات أو نظام أسلحة نووية استراتيجي. ولهذا السبب فمن الواجب بناء نموذج يمثل النظام وتتم الدراسة عليه عوضا عن النظام. ومن الضروري أن يكون النموذج يمثل النظام بشكل دقيق لتكون نتائج التجارب على النموذج مطابقة أو قريبة من الواقع.

4 النموذج الفيزيائي مقابل النموذج الرياضي (Physical Model vs. Mathematical Model)

بالنسبة للعديد من الناس فإن كلمة نموذج توحي لهم بقمرة قيادة منفصلة عن الطائرة تستخدم بغرض التدريب أو مجسم لسفينة كبيرة. هذه أمثلة عن النموذج الفيزيائي (ويدعى أيضا النموذج الأيقوني)، وهو ليس مثالياً للاستخدام من أجل تحليل الأنظمة أو دراسة العمليات. لكن تبين أنه من المفيد بناء النماذج الفيزيائية لدراسة بعض المشاريع هندسيا. لكن معظم الأنظمة والتي تبنى لهذا الغرض(الهندسة) يتم تمثيلها رياضيا على شكل علاقات منطقية وكمية وبعدها يتم معالجتها وتغييرها وملاحظة رد فعل النموذج على التغيرات، وبذلك ما هو رد فعل النظام على هذه التغييرات طبعا إذا كان النموذج صحيح. ربما يكون أبسط مثال عن النموذج الرياضي هي العلاقة $d=rt$ حيث r هي السرعة وt زمن الرحلة وd هي المسافة المقطوعة خلال الرحلة. هذه العلاقة تعطينا نموذجاً صحيحاً (مثلا تحديد موقع مكوك فضاء بعد معرفة السرعة ونقطة البداية والوجهة) لكنه نموذج ضعيف من أجل مهام أخرى (مثل السفر في طريق مزدحم.

5 الطرائق التحليلية مقابل المحاكاة (Analytical solutions vs. Simulation)

حالما نقوم ببناء نموذج رياضي، يجب علينا اختباره لنرى كيف ستكون استجابته لاستفسارات معينة تتعلق بالنظام الذي يمثله. إذا كان النموذج بسيطاً بشكل كاف، فإنه من الممكن أن نعمل على

العلاقات والكميات فيه لنحصل على الحل التحليلي الدقيق. في العلاقة d=rt وإذا كنا نعرف المسافة المراد قطعها والسرعة، فإنه من الممكن العمل على النموذج للحصول على قيمة t حيث t=d/r وهكذا نكون حصلنا على وقت الرحلة. هذا بسيط جدا ويمكن الحصول عليه باستخدام ورقة وقلم، لكن بعض الحلول التحليلية يمكن أن تكون معقدة بشكل كبير جدا، وتتطلب موارد حسابية كبيرة فعلى سبيل المثال ولعكس مصفوفة ببعدين لدينا علاقة رياضية واضحة، لكن الحصول على هذه المصفوفة عدديا هو أمر ليس بهذه البساطة. إذا كان الحل التحليلي للنموذج الرياضي متوفر وكان مجدي حسابيا، فإنه من الأفضل دراسة النموذج بهذه الطريقة بدلا من استخدام المحاكاة. لكن هنالك العديد من الأنظمة المعقدة للغاية وبذلك فإن النماذج الرياضية الخاصة بها معقدة أيضا، ولا يوجد إمكانية للحل التحليلي. في هكذا حالات يجب أن تتم دراسة النموذج بأسلوب المحاكاة.

6 أنواع النماذج

ليكن لدينا نموذج رياضي نريد دراسته باستخدام المحاكاة (لذلك سنشير للنموذج على أنه نموذج المحاكاة) وبذلك علينا أن نبحث عن أدوات محددة للقيام بذلك. إنه من المفيد تقسيم نماذج المحاكاة إلى ثلاثة أبعاد:

6–1 نماذج المحاكاة المتغيرة مقابل الثابتة (Static vs.Dynamic)

نموذج المحاكاة الثابت هو تمثيل للنظام في وقت معين، أو إنه نموذج لنظام مستقل عن الزمن لا يتغير بتغير الزمن. وكمثال عن هذا النوع نماذج مونتيكارلو (Monte Carlo models). وعلى صعيد آخر فإن نموذج المحاكاة المتغير يمثل نظاماً يتغير بتغير الزمن مثل نظام خط تصنيع في مصنع.

6–2 نماذج المحاكاة الحتمية مقابل العشوائية (Deterministic vs. Stochastic)

إذا كان نموذج المحاكاة لا يحوي أي مكونات تعتمد على الاحتمالات (متغيرات عشوائية مثلا) يدعى عندها النموذج بنموذج حتمي(Deterministic) وكمثال عنه نظام مكون من عدة تفاعلات كيميائية. في النماذج الحتمية الخرج يكون محدداً حالما يتم تحديد كميات الدخل والعلاقات في النموذج. بعض الأنظمة يتوجب نمذجتها بحيث تحتوي على مكونات دخل عشوائية، وهذا ما يجعل النموذج عشوائياً. معظم أنظمة الجرد والأرتال تتم نمذجتها لتكون نماذج عشوائية. نماذج المحاكاة العشوائية تنتج خرجاً

عشوائياً بحد ذاته، وبالتالي يجب أن يعالج هذا الخرج على أنه تخمين لحقيقة خواص النموذج، وهذا ما يعد من سلبيات المحاكاة.

3-6 النماذج المستمرة مقابل النماذج المتقطعة (Continuous vs. Discrete)

النظام المتقطع (discrete system) هو نظام تتغير فيه متحولات الحالة بشكل آني في نقاط معينة من الزمن. يعد البنك نظاماً متقطعاً، لأن متحولات الحالة (مثل عدد الزبائن في البنك) يتغير فقط عند دخول زبون جديد إلى البنك أو عندما يغادر الزبون بعد أن يكون قد تم تخديمه.

النظام المستمر (continuous system) هو النظام الذي تتغير فيه متحولات الحالة في كل لحظة زمنية. تعد الطائرة في الجو مثالاً عن الأنظمة المستمرة، لأن متحولات الحالة (مثل الموقع والسرعة) يمكن أن تتغير بشكل مستمر مع الوقت. عدد قليل من الأنظمة تكون متقطعة بشكل كامل أو مستمرة بشكل كامل، لكن ومن أجل معظم الأنظمة هنالك نوع سائد على الآخر، وبذلك يكون من الممكن أن نقسم الأنظمة إلى مستمرة ومتقطعة.

ومن الجدير بالذكر أن القرار في استخدام النموذج المستمر أو المتقطع لنظام معين يعتمد على أهداف خاصة بدراسة هذا النظام، فعلى سبيل المثال، نموذج المرور لطريق سريع يمكن أن يكون متقطعاً إذا كانت خصائص وحركة كل سيارة ضرورية. وإذا كان من الممكن التعامل مع السيارات بالطريقة نفسها فمن الممكن استخدام نموذج مستمر.

7 محاكاة الأحداث المتقطعة (Discrete event simulation)

تعني محاكاة الأحداث المتقطعة أن النظام يتغير فقط عند عدد معين من اللحظات الزمنية. هذه اللحظات الزمنية هي لحظات وقوع الأحداث، إذ يعرف الحدث (event) على أنه حادثة آنية يمكن أن تغير في حالة النظام. ويمكن أن يتم إنجاز محاكاة الأحداث المتقطعة بواسطة الحسابات اليدوية لكن كمية البيانات الواجب تخزينها ومعالجتها بالنسبة لمعظم الأنظمة الواقعية هي التي توجب إنجاز محاكاة الأحداث المتقطعة على الحاسب.

مثال: ليكن لدينا وسيلة تخديم مع مخدم واحد (مثلا حلاق أو مكتب استعلامات في المطار) ونريد أن نخمن متوسط زمن التأخير في رتل الانتظار بالنسبة للزبائن المنتظرة، ويعرف التأخير في الرتل على أنه الزمن الفاصل بين لحظة وصول الزبون واللحظة التي يبدأ فيها بتلقي الخدمة. ومن أجل تخمين معدل التأخير في الرتل سنعتبر أن متحولات الحالة لنموذج محاكاة الأحداث المتقطعة الخاص بالخدمة

هي حالة المخدم (هل المخدم مشغول أم لا وعدد الزبائن التي تنتظر في الرتل ووقت وصول كل زبون). حالة المخدم ضرورية لتحديد إمكانية تخديم الزبون حال وصوله في حال كان المخدم غير مشغول أو يتوجب إضافة الزبون إلى نهاية رتل الانتظار في حال كان المخدم مشغولاً. عندما ينتهي المخدم من تخديم الزبون، فإن عدد الزبائن في الرتل سيستخدم لتحديد هل سيصبح المخدم عاطلاً عن العمل أم أنه سيقوم بتخديم أول زبون في الرتل. يستخدم وقت وصول الزبون لحساب التأخير في الرتل، والذي يساوي الوقت الذي يبدأ فيه الزبون بتلقي الخدمة (وهذا الوقت سيكون معلوم) مطروح منه وقت وصول الزبون. هنالك نوعين من الأحداث في هذا النظام: وصول الزبون وانتهاء تخديم الزبون، والذي يؤدي إلى مغادرة الزبون. يعد الوصول حدث لأنه أدى لتغيير متحولات الحالة الخاصة بالمخدم فالمتحول server status يمكن أن يتغير من idle إلى busy أو أن المتحول المعبر عن عدد الزبائن في الرتل سيزداد بمقدار واحد. وبشكل مشابه فإن المغادرة تعد حدث لأنها تأثر على حالة المخدم من busy إلى idle أو أن عدد الزبائن في الرتل سينقص بمقدار واحد.

كان لكلا الحدثين في–في المثال السابق–تأثير على حالة النظام، لكن في بعض نماذج محاكاة الأحداث المتقطعة يتم استخدام الأحداث لأغراض لها تأثير مختلف عن تأثر الأحداث في هذا المثال، كأن يقوم الحدث بجدولة حدث لعملية في النظام في وقت معين أو أن يقوم بجدولة حدث إنهاء المحاكاة أو لن يقوم الحدث بجدولة أي شيء. وهذا هو السبب الذي يدفعنا للقول بأن الحدث يمكن أن يسبب تغييراً في حالة النظام.

7–1 طرائق تقديم الوقت (Time–Advance Mechanisms)

بسبب الطبيعة المتغيرة لنماذج محاكاة الأحداث المتقطعة، فإنه يجب أن نبقي أثراً للقيمة الحالية لوقت المحاكاة أثناء تقدم عملية المحاكاة، ونحن بحاجة إيضاح إلى آلية لتقديم وقت المحاكاة من قيمة إلى أخرى. نسمي المتحول الذي يعطينا قيمة وقت المحاكاة في نموذج المحاكاة بساعة المحاكة (simulation clock). وحدة الوقت في ساعة المحاكاة لا يتم ذكرها بشكل صريح عندما يتم كتابة النموذج في لغات متعددة الأغراض وتعد وحدة الوقت هي نفسها وحدة متحولات الدخل. وأيضا لا يوجد علاقة بين وقت المحاكاة والوقت الذي يحتاجه النموذج ليتم تنفيذه على الحاسب.

هنالك طريقتان–تاريخياً–لتقديم ساعة المحاكاة:

- التقديم إلى وقت الحدث التالي (Next event time advance)
- تقديم الوقت بزيادات ثابتة (fixed–increment time advance)

وتعد الطريقة الثانية حالة خاصة من الطريقة الأولى.

يتم تهيئة ساعة المحاكاة بالقيمة صفر في طريقة تقديم الوقت إلى وقت الحدث التالي كما يتم تحديد أوقات الأحداث المستقبلية حسب منطق المحاكاة، بعدها يتم تقديم الساعة إلى وقت حدوث أقرب حدث من الأحداث المستقبلية، في تلك اللحظة يتم تعديل حالة النظام للتعبير عن وقوع الحدث ويتم التعديل على أوقات الأحداث المستقبلية. بعد ذلك يتم تقديم ساعة المحاكاة إلى وقت أقرب حدث ويتم تعديل حالة النظام والأحداث المستقبلية وهكذا. هذه العملية في تقديم ساعة المحاكاة من وقت حدث إلى آخر تستمر حتى يتحقق شرط التوقف. ولأن حالة النظام يتم تعديلها عند لحظات وقوع الأحداث في نموذج محاكاة الأحداث المتقطعة فإن فترات الخمول والتي تكون فيها حالة النظام ثابتة (بين وقوع حدثين) يتم تخطيها من خلال تقديم الساعة من وقت حدث لوقت حدث آخر.

مثال:

سنقوم بهذا المثال بتوضيح طريق تقديم الوقت للحدث التالي من أجل مثال نظام الأرتال بمخدم واحد.

نحن بحاجة لتوضيح ما يلي:

t_i هو وقت الوصول للزبون رقم i علما أنه $(t_0=0)$

$A_i=t_i - t_{i-1}$ وهي الوقت الفاصل بين وصول الزبون رقم $i-1$ والزبون رقم i.

S_i هو الوقت الذي يحتاجه المخدم لتخديم الزبون رقم i (بغض النظر عن مقدار التأخر في رتل الانتظار).

D_i التأخر في رتل الانتظار بالنسبة للزبون رقم i.

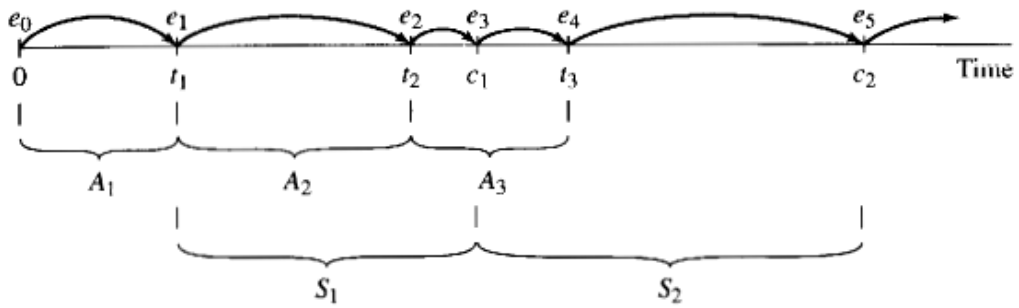
$C_i = t_i + d_i + s_i$ هو الوقت الذي يحتاجه الزبون للمغادرة بعد إنهاء الخدمة.

e_i هو وقت حدوث الحدث رقم i (و يمكن اعتبار $e_0=0$).

إن كل مقدار من المقادير السابقة بشكل عام سيتم تمثيلها على شكل متحولات عشوائية. ولنفرض أن التوزيع الاحتمالي لأوقات الوصول $A_1,A_2,\ldots$ وأوقات التخديم $S_1,S_2,\ldots$ معروفة وتملك تابع توزع تراكمي يشار إليه بالرموز F_A و F_S على التوالي. (بشكل عام ولتحديد F_A و F_S فعلينا جمع البيانات من النظام وحساب التوزع الثابت لهذه البيانات). في اللحظة $e_0=0$ يكون المخدم في حالة عدم انشغال، والوقت t_1 الخاص بأول وصول يتم تحديده من خلال توليد A_1 من التابع F_A وبعدها يتم إضافته إلى 0. بعدها يتم تقديم ساعة المحاكاة من اللحظة e_0 إلى وقت الحدث التالي $e_1=t_1$، في الشكل (2) الأسهم المنحنية تمثل تقديم ساعة المحاكاة. اذا وصل الزبون الأول في اللحظة الزمنية t_1 ووجد المخدم غير مشغول فإنه يتلقى الخدمة مباشرة بحيث يكون وقت انتظاره في الرتل $D_1=0$ وتتغير

حالة المخدم إلى مشغول (idle–>busy). بالنسبة للوقت c_1 فإنه يتم حسابه من خلال توليد S_1 من التابع F_S وتضاف قيمته إلى t_1 وذلك عندما ينتهي الزبون من تلقي الخدمة. وأخيرا فإن وقت الوصول الثاني t_2 يتم حسابه بالشكل $t_2=t_1+A_2$ حيث يتم توليد A_2 من F_A. إذا كان $t_2<c_1$ فإنه يتم تقديم ساعة المحاكاة من e_1 إلى وقت حدوث الحدث الثاني $e_2=t_2$ (إذا كانت c_1 هي الأصغر فإنه يتم تقديم الوقت من e_1 إلى c_1). ولأن الزبون الوارد في اللحظة الزمنية t_2 وجد المخدم في حالة انشغال، بالتالي عدد الزبائن في رتل الانتظار سيزداد بمقدار واحد وسيصبح واحد. إن وقت التخديم الخاص بالزبون الثاني S_2 لن يتم توليده في هذه اللحظة. ويتم حساب وقت الوصول الثالث بالشكل $t_3=t_2+A_3$. حيث كان $c1<t3$ كما هو مبين في الشكل (2)، وبذلك يتم تقديم ساعة المحاكاة من e2 إلى وقت الحدث التالي $e3=c1$ إذ ينتهي الزبون الأول من الخدمة، الزبون الموجود في الرتل (في هذه اللحظة لا يوجد إلا زبون واحد وصل في اللحظة t_2) يبدأ بتلقي الخدمة ويتم حساب تأخيره بالرتل ووقت إنهاء تخديمه بالشكل:

$D_2 = c_1 - t_2$ و $c_2 = c_1 + S_2$ (حيث يتم حساب S_2 من F_S)، وعدد الزبائن المنتظرة في الرتل يتناقص بمقدار واحد ليصبح صفراً. إذا كان $t_3<c_2$ فإنه يتم تقديم ساعة المحاكاة من e_3 إلى وقت الحدث التالي $e_4=t_3$ وهكذا. المحاكاة يمكن أن تنتهي عندما تصل فترة انتظار بعض الزبائن في رتل الانتظار لقيمة معينة.

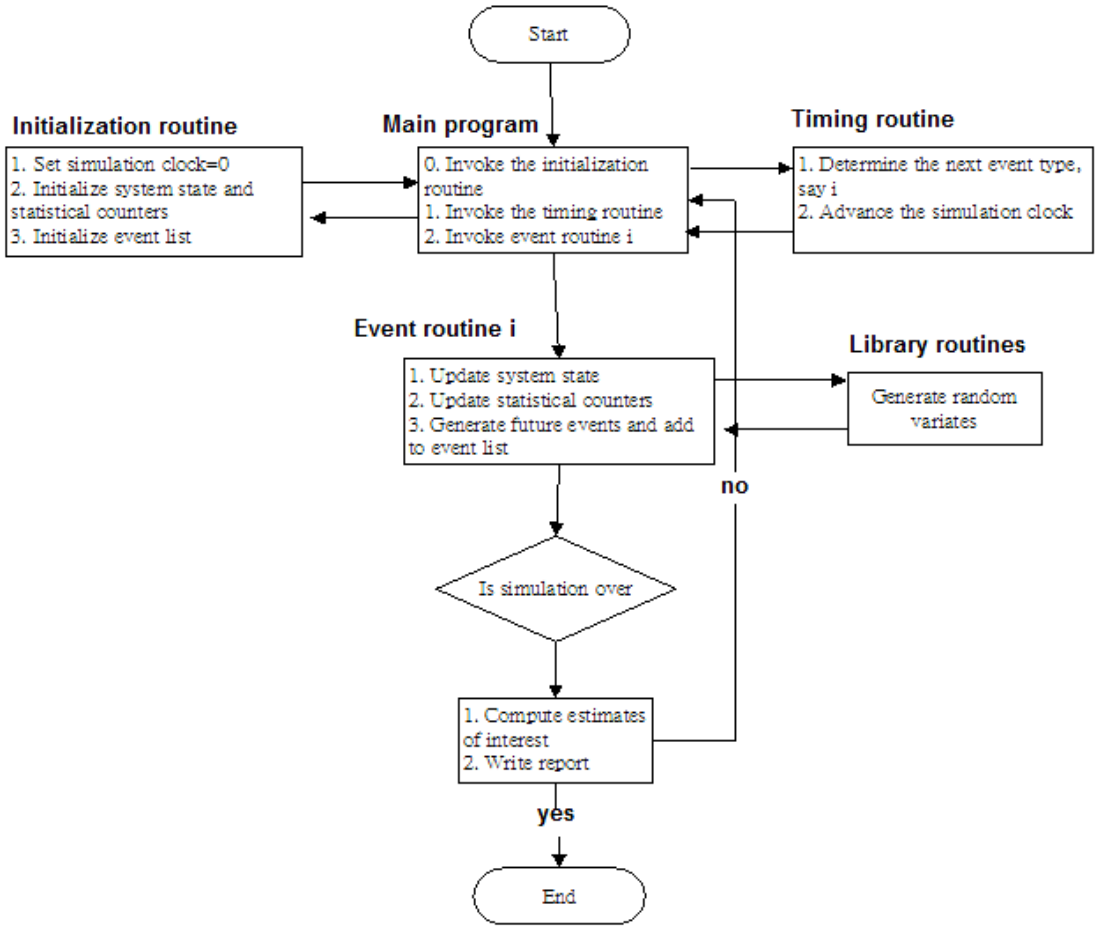


الشكل (2) طريقة تقديم الوقت إلى الحدث التالي من خلال توضيح نظام أرتال بمخدم واحد

2–7 مكونات نموذج الأحداث المتقطعة وتنظيمها

إن نظم الأحداث المتقطعة تتشارك بعدد من المكونات العامة وذلك على الرغم من تطبيقها على تشكيلة واسعة من الأنظمة الواقعية وهنالك منطق لتنظيم هذه المكونات وهذا ما يشجع على البرمجة وتصحيح الأخطاء والتغيير المستقبلي لبرامج المحاكاة. إن المكونات التالية سوف تكون موجودة بشكل أساسي في نماذج محاكاة الأحداث المتقطعة والتي تستخدم طريقة التقديم لوقت الحدث التالي:

- حالة النظام (System state): مجموعة من متغيرات الحالة التي تقوم بوصف النظام في لحظة زمنية معينة.
- ساعة المحاكاة (simulation clock): متحول يعطينا القيمة الحالية لوقت المحاكاة.
- قائمة الأحداث (Event list): قائمة تحوي الوقت التالي الذي يقع فيه كل حدث.
- العدادات الإحصائية (Statistical counters): متحولات تستخدم لتخزين معلومات إحصائية عن أداء النظام.
- دالة التهيئة (Initialization routine): دالة فرعية لتهيئة نموذج المحاكاة في اللحظة 0.
- دالة التوقيت (Timing routine): برنامج فرعي يستخدم لتحديد الحدث التالي في قائمة الأحداث والواجب تنفيذه وبعدها يتم تقديم وقت المحاكاة إلى وقت حدوث الحدث التالي.
- دالة الحدث (Event routine): برنامج فرعي يقوم بالتعديل على حالة النظام عند وقوع حدث من نوع معين (هنالك تابع حدث خاص بكل نوع من الأحداث).
- دوال المكتبة (Library routines): مجموعة من البرامج الفرعية والتي تستخدم لتوليد الأعداد العشوائية من توزيعات احتمالية عديدة.
- مولد التقارير (Report generator): برنامج فرعي يقوم بتخمين (من العدادات الإحصائية) مقاييس معينة للأداء ويقوم بإنتاج تقارير عندما تنتهي المحاكاة.
- البرنامج الرئيس (Main program): برنامج فرعي يقوم باستدعاء تابع التوقيت لتحديد الحدث التالي وبعدها يعطي التحكم لتابع الحدث المناسب من أجل التعديل المناسب على حالة النظام. يمكن للبرنامج الرئيس أن يختبر إنهاء المحاكاة وأن يستدعي مولد التقارير عند انتهاء المحاكاة.



الشكل (3) المخطط التدفقي للتحكم بطريقة التقديم إلى وقت الحدث التالي

الشكل (3) يبين العلاقات المنطقية بين مكونات نموذج محاكاة الأحداث المتقطعة. تبدأ المحاكاة في اللحظة 0 عندما يقوم البرنامج الرئيسي باستدعاء إجرائية التهيئة، حيث يتم إعطاء ساعة المحاكاة القيمة 0 ويتم تهيئة حالة النظام والعدادات الإحصائية، ويتم تهيئة قائمة الأحداث. بعد أن يعود التحكم للبرنامج الرئيس يتم استدعاء إجرائية التوقيت لتحديد أي نوع من الأحداث على وشك الحصول. إذا كان الحدث التالي من النوع i يتم تقديم ساعة المحاكاة إلى وقت الحدث i الذي على وشك الحصول وبعدها يعود التحكم إلى البرنامج الرئيسي. بعدها يقوم البرنامج الرئيس باستدعاء إجرائية الحدث i، وعادة ما يحصل ثلاث فعاليات:

- (1) يتم تعديل حالة النظام بما يناسب وقوع الحدث i
- (2) يتم جمع معلومات عن أداء النظام من خلال التعديل على العدادات الإحصائية
- (3) يتم توليد وقت حدوث الحدث التالي

وهذه المعلومة يتم إضافتها إلى قائمة الأحداث. عادة ما يكون من الضروري توليد قيم عشوائية من توزيع احتمالي معين من أجل تحديد أوقات الأحداث المستقبلية، سنشير لمولدات القيم العشوائية بمصطلح التشكيل العشوائي (random variate). بعد انتهاء كل عمليات المعالجة، سواء في إجرائية الحدث i أو في البرنامج الرئيس، يتم فحص شرط التوقف للمحاكاة، وذلك لتحديد وجوب انتهاء المحاكاة أم عدم وجوب ذلك. في حال وجب إنهاء المحاكاة، فإنه يتم استدعاء مولد التقارير من قبل البرنامج الرئيس من أجل حساب التوقع (من العدادات الإحصائية) لمقاييس أداء معينة ومن أجل توليد التقارير. في حال لم يكن من الواجب إنهاء المحاكاة فإن التحكم يعود إلى البرنامج الرئيس وتتكرر العملية حتى يتحقق شرط التوقف.

إن تنظيم برنامج محاكاة الأحداث المتقطعة باستخدام طريقة تقديم الوقت إلى وقت الحدث التالي والتي تم شرحها في ما سبق تعد مثالية بشكل كبير عند استخدام لغات البرمجة متعددة الأغراض (java,c#,c++ ...) إن هذه الطريقة تدعى بطريقة جدولة الأحداث من أجل محاكاة النموذج (event–scheduling approach) لأن أوقات الأحداث المستقبلية يتم التصريح عنها في البرنامج بشكل صريح ويتم جدولتها ليتم محاكاتها في المستقبل.

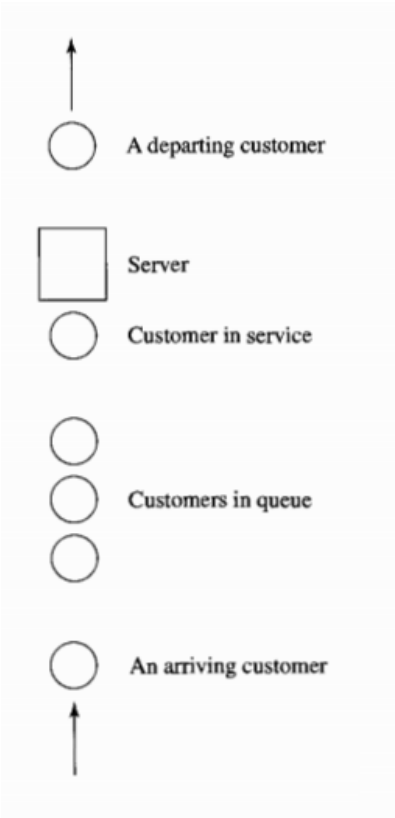
3–7 محاكاة رتل بمخدم واحد

نص المسألة (problem Statement)

ليكن لدينا نظام انتظار بمخدم واحد كما هو موضح في الشكل(4) بحيث تكون A1,A2,... متحولات عشوائية تتبع توزيع مستقل ومتماثل((Independent and identically distributed (IID) وتمثل الفترات الزمنية .

المقصود بالتوزيع المتماثل (Identically distributed) أي أن الفترات الزمنية تملك التوزيع الاحتمالي نفسه. الزبون الذي يصل ويجد المخدم عاطل عن العمل يدخل إلى الخدمة بشكل مباشر، وأوقات التخديم S1,S2,... للزبائن هي متحولات عشوائية من نوع (IID) وهي مستقلة عن الفترات الزمنية. الزبون الذي يصل ويجد المخدم مشغول ينضم إلى رتل الانتظار الوحيد. حتى تنتهي فترة تخديم الزبون الموجود على المخدم. عندما ينتهي الزبون من التخديم فإنه يغادر ليقوم المخدم باختيار زبون آخر من الرتل علما أن رتل الانتظار من النوع الداخل أولا خارج أولا (first in first out). تعد المحاكاة في حالة غير فعالة إذا لم يدخل زبائن وكان المخدم غير مشغول.

في اللحظة 0 سنبدأ بانتظار وصول أول زبون، والذي يمكن أن يأتي بعد أول فترة زمنية A1 (قد يأتي الزبون في اللحظة 0 لكن هذه يعتمد على النموذج).



الشكل(4) نظام انتظار بمخدم واحد

سوف نقوم بمحاكاة النظام حتى ينهي عدداً محدداً n من الزبائن فترة انتظارهم في الرتل (أي أن المحاكاة ستنتهي بعد انتهاء المخدم من تخديم الزبون رقم n). من الجدير بالملاحظة أن وقت نهاية المحاكاة هو متحول عشوائي، لأنه يعتمد على قيم الفترات الزمنية وعلى أوقات التخديم وهذه القيم هي متغيرات عشوائية.

من أجل قياس أداء مثل هذا النظام سوف نقوم بتخمين ثلاث كميات.

أولا: معدل التأخير المتوقع في الرتل من أجل n زبون انهوا فترات انتظارهم في الرتل، سنرمز لهذه الكمية بـ d(n). إن كلمة (المتوقع) الواردة في تعريف d(n) تعني: من أجل تشغيل معين للمحاكاة (أو في الحالة التي يتم فيها تشغيل النظام الواقعي الممثل بنموذج المحاكاة) يكون معدل التأخير الفعلي الملاحظ من أجل n زبون يعتمد على قيم المتحولات العشوائية التي تحدد قيم الفترات الزمنية وفترات التخديم، ومن أجل تشغيل آخر للمحاكاة (أو من أجل يوم آخر للنظام الواقعي) فإن الزبائن يمكن أن تصل في أوقات مختلفة، ووقت التخديم لكل زبون يمكن أن يختلف، وبذلك الحصول على قيم مختلفة

لمعدل التأخير من أجل n زبون. وبذلك فإن معدل التأخير من أجل تشغيل معين للمحاكاة يمكن أن يعد متحولاً عشوائياً بحد ذاته. ما نريد تخمينه، d(n) هو القيمة المتوقعة لهذا المتحول العشوائي، بما أن d(n) هو عبارة عن معدل تأخير n زبون فهو يعطى بالعلاقة

$$\hat{d}(n) = \frac{\sum_{i=1}^{n} D_i}{n}$$

و التي تمثل معدل n تأخير تمت ملاحظتهم في المحاكاة. تم إضافة الرمز ^ لنعبر على أن هذه القيمة يتم تخمينها. من الجدير ملاحظته بالنسبة للتأخير إنه يمكن أن يأخذ القيمة 0 في حال وصول الزبون وكان النظام في حالة غير مشغولة (empty and idle). مع هذا النموذج يمكن أن يكون D1=0 وإذا كان هنالك العديد من قيم التأخير الصفرية فهذا يدل على أن النظام يقدم خدمة جيدة. وعلينا أن نقيس ذلك ونبينه في خرج المحاكاة.

ثانيا: إن القيمة التخمينية لـ(d(n تعطي معلومات عن أداء النظام من وجهة نظر الزبائن، لكن يمكن أن يكون مدير النظام يريد معلومات مختلفة. في الحقيقة ولأن معظم أنظمة المحاكاة الواقعية معقدة نوعاً ما ويمكن أن يكون تشغيلها مستهلكاً للوقت فإنه يتم جمع عدة مقاييس للأداء وإظهارها في الخرج، هذه المقاييس تبين جوانب مختلفة لسلوك النظام. أحد هذه المقاييس وبالنسبة للنموذج البسيط الذي نقوم بدراسته هو q(n) أي معدل عدد الزبائن (الغير مخدمة) والتي تنتظر في الرتل، حيث n ضروري لتبين أن المعدل تم أخذه خلال الفترة الزمنية اللازمة لملاحظة nزبون في الرتل والتي تعرف شرط التوقف. هذا معدل من نوع آخر مختلف عن معدل التأخر في الرتل d(n)، لأنه يتم أخذه بشكل مستمر مع الزمن وليس مع كل زبون (بشكل متقطع). لذلك نحن بحاجة لتعريف المقصود من وراء معدل عدد الزبائن في الرتل.

ليكن Q(t) يمثل عدد الزبائن في الرتل في اللحظة الزمنية t، من أجل أي عدد حقيقي t>=0 ، وليكن T(n)هو الوقت اللازم لملاحظة n تأخر في الرتل. بالتالي ومن أجل أي لحظة زمنية t تقع ضمن المجال [0,T(n)] سيكون Q(n) عبارة عن عدد صحيح موجب. وليكن p_i نسبة توقع أن يكون Q(t)=i في اللحظة الزمنية t(علما أن قيمة p_i تقع ضمن المجال [0,1]) .

و بذلك يمكن تعريف q(n) من العلاقة الآتية.

$$q(n) = \sum_{i=0}^{\infty} ip_i$$

أي أن q(n) هو معدل موزون لقيم i الممكنة والتي تعبر عن طول الرتل Q(t)، الأوزان تعبر عن الوقت المحتمل للرتل ليبقى بطول محدد. من أجل تخمين q(n) من المحاكاة ، علينا فقط أن نبدل كل p_i بقيمة التخمين الخاصة بها .

$$\hat{q}(n) = \sum_{i=0}^{\infty} i\hat{p}_i \quad 1\text{-}7$$

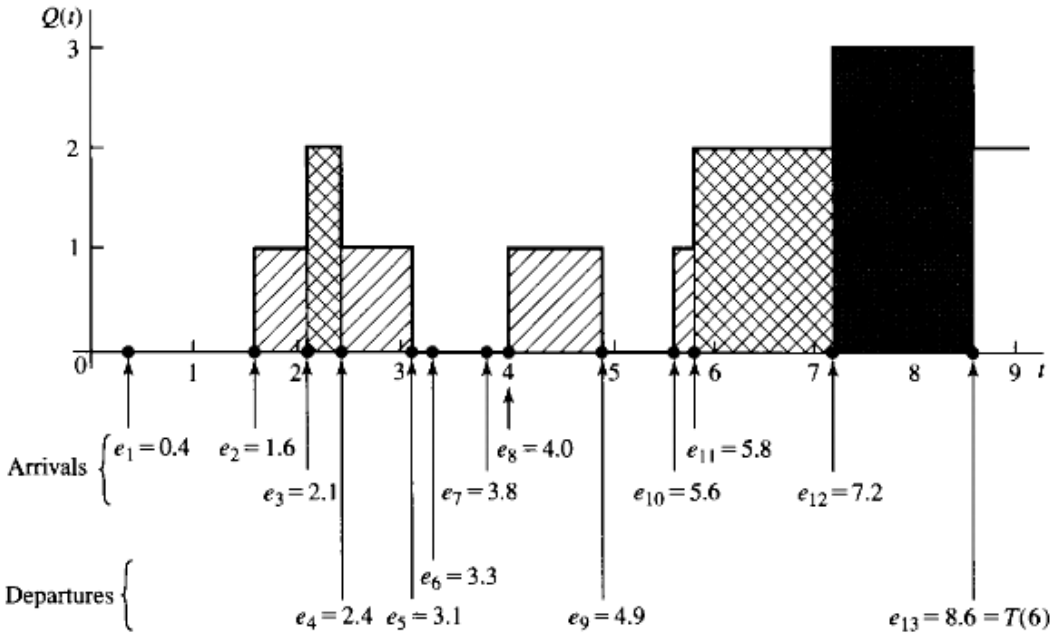
إذ $\hat{p}_i$ هي نسبة ملاحظة (بدلا من التوقع) وقت المحاكاة عندما يكون عدد الزبائن في الرتل يساوي i. إنه من الأسهل كتابة $\hat{q}(n)$ باستخدام اعتبارات هندسية .ليكن T_i هو عبارة عن الوقت الكلي في المحاكاة والذي يكون فيه طول الرتل يساوي i .بذلك يكون :

$T(n) = T_0 + T_1 + T_2 + \ldots$ and $\hat{p}_i = T_i/T(n)$

و بذلك يمكن أن نكتب العلاقة (7–1) بالشكل :

$$\hat{q}(n) = \frac{\sum_{i=0}^{\infty} iT_i}{T(n)} \quad 2\text{-}7$$

الشكل (5) يبين المسارات الزمنية الممكنة لـ Q(t) من أجل هذا النظام وفي حالة n=6. أوقات الوصول هي 0.4,1.6,2.1,3.8,4.0,5.6,5.8,7.2 ، وأوقات المغادرة (انتهاء التخديم) هي : 2.4,3.1,3.3,4.9,8.6 ،و تنتهي المحاكاة في اللحظة T(6)=8.6.



الشكل (5) Q(t) وأوقات الوصول ،أوقات المغادرة في نظام الانتظار بمخدم واحد .

بالنظر إلى الشكل (5) فإنه لا يتم حساب الزبائن الذين يتلقون الخدمة (إن وجدوا)، إذا هنالك زبون واحد في النظام ويتم تخديمه بين اللحظة الزمنية 0.4 واللحظة 1.6، سيكون الرتل فارغا [Q(t) = 0] والشيء نفسه صحيح بين اللحظة الزمنية 3.1 واللحظة 3.3 وأيضا بين 3.8 و4.0 وأيضا بين 4.9 و5.6. بين اللحظة 3.3 و3.8 يكون النظام فارغا من الزبائن والمخدم يكون غير مشغول، والحالة نفسها تتكرر بين 0 و0.4. لحساب q^(n) علينا أن نقوم أولا بحساب جميع قيم T_i، كفترات بحيث يكون عدد Q(t) فيها مساوي القيمة 0 ثم 1 ثم 2 وهكذا:

$T_0 = (1.6 - 0.0) + (4.0 - 3.1)+(5.6-4.9)=3.2$

$T_1 = (2.1 - 1.6) + (3.1 - 2.4)+(4.9 -4.0)+(5.8-5.6)=2.3$

$T_2=(2.4-2.1)+(7.2-5.8)=1.7$

$T_3=(8.6-7.2)=1.4$

$T_i=0$ من أجل كل قيم i>=4 ، لأن طول الرتل لم يتجاوز 3 في مثالنا.

$$\sum_{i=0}^{\infty} iTi = (0 * 3.2) + (1 * 2.3) + (2.* 1.7) + (3 * 1.4) = 9.9$$

العلاقة (3–7)

بالتالي يكون القيمة التخمينية لمعدل وقت الانتظار في الرتل ومن أجل تنفيذ معين للمحاكاة

q^(6)=9.9/8.6=1.15 . الآن يمكن ملاحظة أن كل حد غير صفري في الطرف الأيمن للعلاقة (7–3) مرتبط بمنطقة مظللة في الشكل (5) :1*2.3 ممثل بالمناطق المظللة بشكل قطري (4 مناطق) ، 2*1.7 المناطق المظللة بخطوط متقاطعة (منطقتين)، 3*1.4 هي المنطقة المظللة بالأسود (منطقة واحدة).بعبارة أخرى، إن مجموع البسط في العلاقة (7–2) هو عبارة عن المساحة الواقعة تحت منحني Q(t) والمحصورة بين بداية ونهاية المحاكاة. ونحن نعلم إن المساحة تحت المنحني هي عبارة عن تكامل، ويمكننا كتابة العلاقة بالشكل الآتي.

$$\sum_{i=0}^{\infty} iT_i = \int_0^{T(n)} Q(t)\,dt$$

و يمكن التعبير عن تخمين q(n) بالشكل :

$$\hat{q}(n) = \frac{\int_0^{T(n)} Q(t)\,dt}{T(n)} \qquad 4\text{-}7$$

إن العلاقة (7–4) والعلاقة (7–2) متكافئتان والعلاقتين تستخدمان لحساب q^(n).

لكن يفضل استخدام العلاقة (7–4) لأن التكامل في هذه العلاقة يمكن حسابه كمجموع مساحات المستطيلات وذلك أثناء تنفيذ عملية المحاكاة وهذا أسهل من حل العلاقة (7–2) بشكل صريح. إضافة إلى ذلك فإن العلاقة (7–4) تعطينا المعدل المستمر لـ Q(t) لأن التكامل يمكن اعتباره على أنه مجموع مستمر.

ثالثا : آخر قياس لأداء النظام يمكن أن نظهره في الخرج هو مقدار انشغال المخدم. القيمة المتوقعة لاستخدام المخدم(utilization) هي نسبة الوقت المتوقعة خلال المحاكاة والتي يكون فيها المخدم في حالة انشغال (من اللحظة الزمنية 0 إلى اللحظة T(n)) وهي عبارة عن عدد ضمن المجال [0,1] ويرمز لها بالشكل u(n) من أجل تنفيذ معين للمحاكاة، فيكون القيمة التخمينية لانشغال المخدم u^(n) وهي نسبة الوقت الذي تم ملاحظة انشغال المخدم خلاله. الآن يمكن حساب u^(n) بشكل مباشر أثناء زمن تنفيذ المحاكاة من خلال ملاحظة الأوقات التي تتغير فيها حالة المخدم (من مشغول إلى غير مشغول وبالعكس) وإجراء العمليات اللازمة لجمع فترات الانشغال وتقسيمها على الزمن الكلي للمحاكاة. ومن السهل اعتبار هذه الكمية كمعدل مستمر مع الزمن، بشكل مشابه لطول رتل الانتظار، لذلك سنعرف تابع انشغال المخدم (busy function)

$$B(t)=\begin{cases}1 \; if \; the \; setver \; is \; busy \; at \; time \; t \\ 0 \; if \; the \; setver \; is \; idle \; at \; time \; t\end{cases}$$

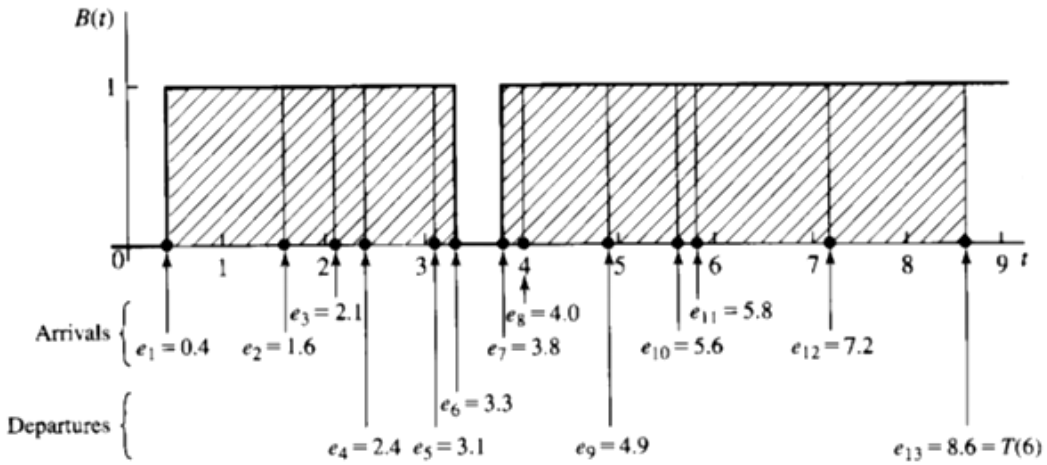
و يمكن حساب u^(n) كنسبة للوقت بحيث يكون B(t) = 1 .الشكل (6) يبين B(t) من أجل نفس مثال المحاكاة الموجود في الشكل (5) .في هذه الحالة سيكون لدينا

$$\hat{u}(n) = \frac{(3.3 - 0.4) + (8.6 - 3.8)}{8.6} = \frac{7.7}{8.6} = 0.90 \qquad 7\text{-}5$$

و هذا ما يبين أن المخدم كان في حالة انشغال بنسبة 90 بالمئة خلال فترة المحاكاة. إن المعادلة (7–5) تكافئ المساحة المحصورة تحت تابع الإنشغال B(t) خلال فترة المحاكاة، لأن ارتفاع B(t) دائما يساوي الصفر أو الواحد.

$$\hat{u}(n) = \frac{\int_0^{T(n)} B(t)\, dt}{T(n)} \qquad 7\text{-}6$$

نلاحظ مرة أخرى أن u^(n) هو معدل مستمر بالنسبة للتابع B(t) . وبشكل مشابه لـq^(n)، فإن سبب كتابة u^(n) على شكل تكامل في العلاقة (7–6) هو سبب حسابي بحت، لأنه مع تقدم تنفيذ المحاكاة فإن تكامل B(t) يمكن أن يتم حسابه بسهولة من خلال جمع مساحة المستطيلات.



الشكل (6) B(t) وأزمنة الوصول وأزمنة المغادرة لنظام الانتظار بمخدم واحد .

تعد الاستخدامية من الإحصائيات المهمة بالنسبة لنماذج المحاكاة التي تحتوي على أرتال فهي تقدم معلومات مهمة تساعد في تحديد إمكانية وجود عنق زجاجة (إذا كانت الاستخدامين قريبة من مئة

بالمائة ومقرونة بازدحام كبير في الرتل) وتساعد أيضا في تبيان السعوية المفرطة (الاستخدامية المنخفضة) وهذا مفيد إذا كان المخدم مكلف جدا.

إذا مقاييس الأداء التي تحدثنا عنها هي: معدل الانتظار في الرتل d^(n) ومعدل عدد الزبائن المنتظرة في الرتل بالنسبة للزمن q^(n) ونسبة انشغال المخدم u^(n).إن معدل الانتظار في الرتل هو مثال عن إحصائية متقطعة زمنية ، لأنها تعرف بالاعتماد على مجموعة من المتغيرات العشوائية {Di} والتي لديها دليل زمني متقطع أي i={1,2,3,..}. إن معدل عدد الزبائن المنتظرة في الرتل بالنسبة للزمن ونسبة استخدام المخدم هي أمثلة عن إحصائيات مستمرة مع الزمن، لأنها تعرف بالاعتماد على متحولات عشوائية {Q(t)} و{B(t)}،و كل واحد من هذه المتحولات لديه دليل زمني مستمر بحيث t>=0.

هذا النموذج يحتوي على حدث وصول الزبون وحدث المغادرة (بعد التخديم) ومتحولات الحالة الضرورية لتخمين q^(n) وd^(n) وu^(n) هي :

حالة المخدم (0 إذا كان غير مشغول و1 إذا كان مشغولاً)

- عدد الزبائن في الرتل
- وقت الوصول لكل زبون في الرتل حاليا
- وقت آخر حدث

وقت آخر حدث ضروري لتحديد مساحة المستطيلات التي تعبر عن القيم التخمينية لـ q^(n) وu^(n)

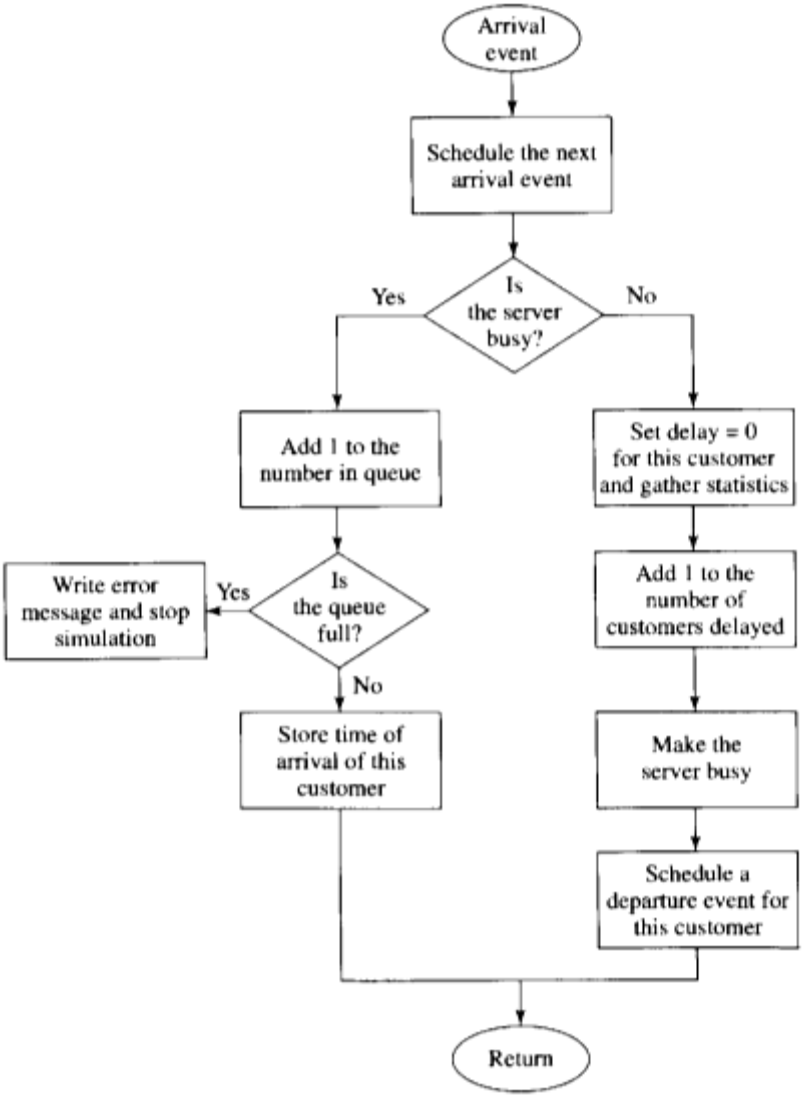
.

ترتيب برنامج المحاكاة لنموذج الانتظار بمخدم واحد

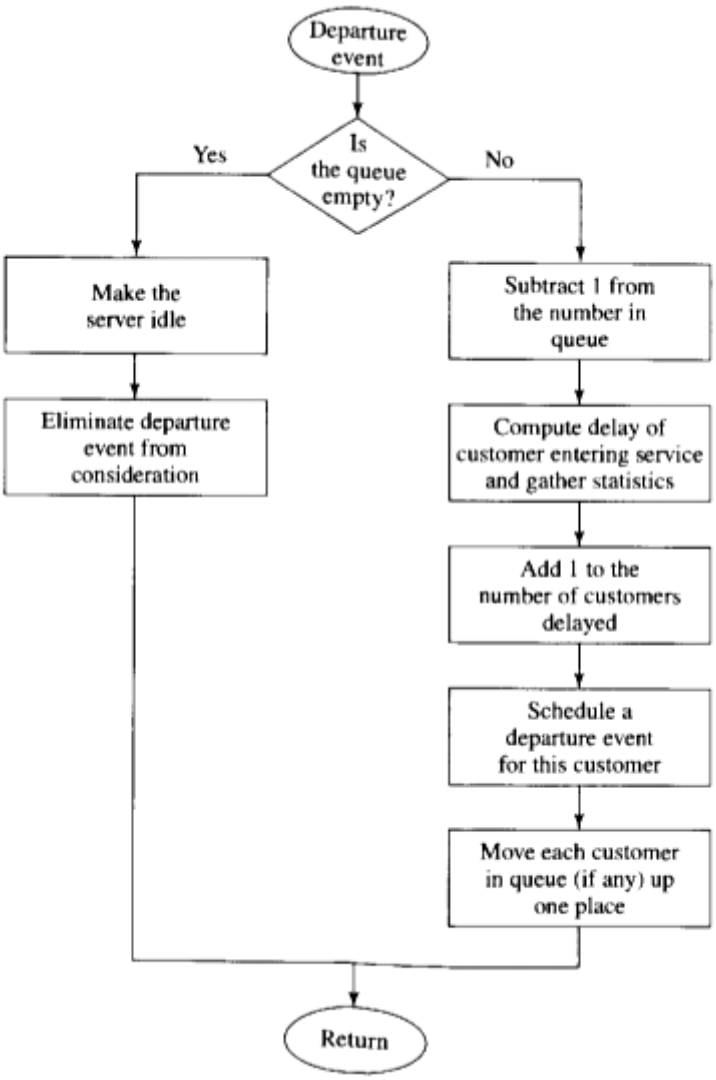
إن هذا النموذج يحتوي على حدثين هما:

- حدث وصول الزبون الشكل (7)
- حدث مغادرة الزبون (8)

ويتم التغيير على المتحولات الإحصائية تبعا للأحداث التي يتم تنفيذها.



الشكل (7) المخطط التدفقي لإجرائية الوصول في نموذج الانتظار بمخدم واحد



الشكل (8) المخطط التدفقي لإجرائية المغادرة في نموذج الانتظار برتل واحد

الملحق(2)

مثال عن المقابلة و الاختزال (بيانات الطقس) [2]

ليكن لدينا عدد من حساسات الطقس التي تقوم بجمع بيانات عن حالة الطقس من أجل كل ساعة وهذه الحساسات منتشرة عبر العالم، كما أنها تقوم بجمع كمية كبيرة من البيانات، وهذه الحالة تعتبر مناسبة للتحليل باستخدام المقابلة والاختزال لأننا نريد أن نعالج كل البيانات، كما أن البيانات هي موجهة بالسجلات (record–oriented) وهي نصف مهيكلة (semi–structured).

1 صيغة بيانات الطقس

إن مصدر هذه البيانات هو مركز بيانات المناخ الوطني الأمريكي(NCDC). إن هذه البيانات مخزنة بصيغة ASCII بشكل سطري وكل سطر يمثل سجل واحد. كل سطر يحتوي على عدد من القياسات التي قد تكون اختيارية أو تحتوي على قيم متغيرة. من أجل التسهيل سنهتم بالقياسات الأساسية مثل درجة الحرارة لأنها موجودة في كل سجل وتخزن بطول ثابت.

الشكل (1) يبين مثالا عن سجل سطري، ومن أجل التوضيح تم كتابة هذا السطر على عدة سطور لإضافة بعض التعليقات لتوضيح دلالة القيم.

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

الشكل (1) صيغة سجل يمثل بيانات الطقس بالنسبة للمركز الوطني الأمريكي لبيانات الطقس

إن ملفات البيانات يتم تنظيمها تبعا للتاريخ ولمحطات الطقس. بحيث يكون هنالك مجلد لكل عام ابتداء من العام 1901، وكل مجلد يحتوي على ملفات من نوع gz وكل ملف يحتوي قراءات محطة طقسية لعام كامل. والشكل (2) يبين عدد من الملفات الخاصة بعام 1990.

هنالك الآلاف من المحطات الطقسية، بالتالي مجموعة البيانات تتألف من عدد كبير من الملفات الصغيرة. بشكل عام إنه من الأفضل من ناحية الأداء معالجة عدد أقل من الملفات ذات الحجم الأكبر لذلك يمكن أن يتم دمج ملفات كل عام إلى ملف واحد.

```
% ls raw/1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
010030-99999-1990.gz
010040-99999-1990.gz
010080-99999-1990.gz
010100-99999-1990.gz
010150-99999-1990.gz
```

الشكل(2) عدد من الملفات الطقسية الخاصة بالعام 1990

2 معالجة البيانات باستخدام Hadoop

من أجل الاستفادة من المعالجة التفرعية التي يقدمها Hadoop، سنقوم بشرح الاستعلام الذي نريد تنفيذه كعمل من نوع المقابلة والاختزال.

2–1 المقابلة والاختزال

إن أعمال المقابلة والاختزال تقوم بتقسيم المعالجة إلى مرحلتين:

3– مرحلة المقابلة Map phase.

4– مرحلة الاختزال Reduce phase.

وكل مرحلة تحتوي على دخل وخرج بشكل مفتاح–قيمة، وإن نوع المفاتيح أو القيم يتم اختياره من قبل المبرمج. وعلى المبرمج أن يحدد دالتين هما: دالة المقابلة ودالة الاختزال.

إن دخل مرحلة المقابلة هو البيانات الخام الخاصة بـ NCDC. سنعتبر بيانات الدخل كبيانات نصية بحيث نعالج كل سطر بشكل منفصل بحيث يكون السطر يمثل القيمة أما المفتاح هو رقم السطر، سيتم تجاهل المفتاح فيما بعد لعدم الحاجة له.

إن تابع المقابلة في هذا المثال هو تابع بسيط. حيث سنقوم باستخراج العام ودرجة الحرارة (القيم التي نهتم بها). هي هذه الحالة يعتبر تابع المقابلة كما لو أنه يقوم بمرحلة تجهيز البيانات، بحيث يستطيع تابع الاختزال العمل على البيانات لإيجاد درجة الحرارة الأعلى من أجل كل سنة.

هذا ويعتبر تابع المقابلة مكانا مناسبا لحذف السجلات السيئة: حيث نقوم بترشيح درجات الحرارة الغير مدخلة أو التي تمثل قيم متطرفة أو خاطئة.

ومن أجل تبيان الطريقة التي يعمل بها تابع المقابلة سنقدم المثال التالي حيث يحتوي الشكل (3) عينة من أسطر بيانات الدخل (حيث تم حذف بعض الحقول التي لا نهتم لها من أجل مناسبة الشكل للصفحة)

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

الشكل(3) عينة من أسطر بيانات دخل المقابل

يتم تمثيل هذه الأسطر على شكل أزواج قيمة–مفتاح كما في الشكل (4)

```
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)
```

الشكل(4) بيانات دخل المقابل على شكل أزواج قيمة– مفتاح

إن المفاتيح هي أرقام الأسطر في الملف، علما أننا تجاهلناها في تابع المقابلة الخاص بنا. إن تابع المقابلة يقوم باستخراج درجة الحرارة والسنة (المبينان باللون الأسود الغامق)، ويقوم بإرسال هذه القيم كخرج (تم اعتبار قيم درجة الحرارة كقيم صحيحة): الشكل (2–5) يبين نتيجة تابع المقابلة

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

الشكل (5) نتيجة تابع المقابلة

إن خرج تابع المقابلة يتم معالجته ضمن إطار المقابلة والاختزال قبل البدء بإرساله إلى تابع الاختزال. هذه المعالجة تشتمل على الترتيب وتجميع أزواج قيمة–مفتاح تبعا للمفاتيح. بالتالي ستصل البيانات إلى تابع الاختزال بالشكل التالي.

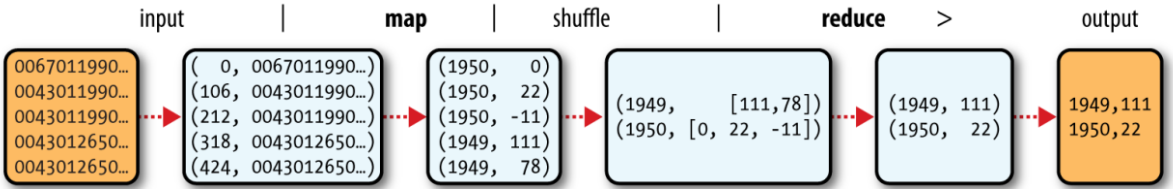
```
(1949, [111, 78])
(1950, [0, 22, -11])
```

كل عام سيظهر مع قائمة تضم كل درجات الحرارة المقروءة. وستكون مهمة تابع الاختزال منحصرة بالمرور عبر القائمة لاختيار درجة الحرارة الأكبر: وسيكون الخرج النهائي حسب الشكل (2–6) الذي يمثل درجة الحرارة العظمة لكل سنة.

```
(1949, 111)
(1950, 22)
```

الشكل (6) نتيجة تنفيذ تابع الاختزال

إن الشكل (7) يمثل التدفق الكلي للبينات.



الشكل (7) التدفق الكلي للبيانات ضمن اطار المقابلة والاختزال

2–2 المقابلة والاختزال في جافا

سنتكلم الآن عن اسلوب عمل برامج المقابلة والاختزال من خلال التحدث عن الشيفرة المصدرية للمثال الخاص ببيانات الطقس. سنحتاج إلى ثلاث قطع من الشيفرة البرمجية هي:

1– تابع المقابلة

2– تابع الاختزال

3– شيفرة مصدرية من أجل تشغيل العمل.

تابع المقابلة

يتم تمثيل تابع المقابلة من خلال الصف Mapper، والذي يقوم بتعريف طريقة مجردة تدعى ()map.

إن الشكل (2–8) يبين تحقيق دالة المقابلة في مثالنا.

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);


      if (airTemperature != MISSING && quality.matches("[01459]")) {
        context.write(new Text(year), new IntWritable(airTemperature));
      }
    }
  }
```

الشكل (8) النص البرمجي لصف الMapper

إن الصف Mapper يعتبر نوع كما أنه يحدد نوع البيانات لمفاتيح الدخل والخرج وقيم الدخل والخرج.

ومن أجل المثال الذي بين يدينا:

1– نمط معطيات مفاتيح الدخل هو long integer

2– نمط معطيات قيم الدخل هو سطر من النص text

3– نمط معطيات مفاتيح الخرج هو نص text (يمثل السنة)

4– نمط معطيات قيم الخرج هي درجة الحرارة integer

إن Hadoop يشتمل على أنماط معطيات خاصة به يمكن مقابلتها بأنماط المعطيات في لغة جافا حيث :

النمط في Hadoop	النمط في جافا
Text	String
LongWritable	Long
IntWritable	Int

الجدول (1) بعض أنماط Hadoop وما يقابلها من أنماط جافا

يتلقى تابع المقابلة المفتاح والقيمة ويقوم هذا التابع بتحويل نص السطر الممر إلى قيمة من نوع String ويستخدم الدالة ()substring للحصول على الحقول التي نريدها.

كما يقوم تابع المقابلة بتأمين مثيل من الصف Context من أجل كتابة الخرج عليه. في هذا المثال قمنا بكتابة السنة كغرض نصي (لأننا استعملناها كمفتاح)، ودرجة الحرارة تم اعتبارها من النوع IntWritable.

تابع الاختزال

بشكل مشابه لتابع المقابلة يتم تعريف ضمن الصف Reducer كما في الشكل(2–9).

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

الشكل (9) النص البرمجي لتابع الاختزال

حيث يشتمل هذا التابع على أربع متحولات ذات أنماط رسمية تحدد أنماط الدخل والخرج، يجب أن تتوافق أنماط خرج تابع المقابلة مع أنماط دخل تابع الاختزال:Text and IntWritable . وفي مثالنا سيكون نمط معطيات خرج تابع الاختزال: Text and IntWritable من أجل السنة وأعظم درجة حرارة فيها، حيث حصلنا على هذه الدرجة نتيجة المرور على درجات الحرارة ومقارنتها مع أعظم درجة حرارة للسجلات التي تم المرور عليها سابقا.

الشيفرة المصدرية لتشغيل عمل المقابلة والاختزال

إن الشكل (10) يبين الشيفرة المصدرية اللازمة لتشغيل مثال المقابلة والاختزال الخاص ببيانات الطقس

```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

الشكل (10) الشيفرة البرمجية اللازمة لتشغيل برنامج المقابلة والاختزال

إن الصف Job يوصف عمل المقابلة والاختزال ويعطينا التحكم في طريقة تنفيذ العمل. علينا أن نقوم بحزم الشيفرة المصدرية بملف من نوع JAR عند تنفيذ العمل على عنقود Hadoop (حيث سيقوم Hadoop بتوزيعه على العنقود). بدلا من التصريح عن أسم ملف الـJAR يمكننا أن نمرر صف العمل من خلال الدالة setJarByClass() حيث سيقوم Hadoop بتحديد ملف الـ JAR المناسب والذي يحتوي على صف العمل.

بعدها علينا أن نحدد مسار الدخل ومسار الخرج. مسار الدخل يتم تحديده من خلال استدعاء الدالة الثابتة addInputPath() على الصف FileInputFormat.مسار البيانات يمكن أن يكون ملف أو مجلد (اذا كان المسار مجلد فسيتم اعتبار كل الملفات ضمنه كملفات دخل) أو أن يكون مسار بيانات

الدخل عبارة عن نمط ملف. كما يمكن أن يتم استدعاء الدالة addInputPath عدة مرات من أجل استخدام دخل من عدة مسارات.

مسار الخرج (هو مسار وحيد) يتم تحديده من استدعاء الدارة ()setOutputPath على الصف FileOutputFormat. حيث يحدد هذا المسار المجلد الذي سيحتوي على ملفات الخرج التي ستنتج عن دالة الاختزال.

بعد ذلك يتم تحدي نوع عملية المقابلة ونوع عملية الاختزال من خلال الدوال ()setMapperClass and setReducerClass().

كما أن الدوال ()setOutputValueClass and ()setOutputKeyClass يتحكمان بنوع خرج تابع الاختزال.

الدوال ()setMapOutputValueClass and ()setMapOutputKeyClass يتحكمان بنوع خرج تابع المقابلة.

علما أنه ليس هنالك ضرورة لتحديد نمط خرج المقابلة لان وعلى الأغلب يمتلك نفس نمط دخل الاختزال.

بعد ذلك يتم استدعاء الدالة waitForCompletion التي تقوم بتسليم العمل وتنتظر انتهاء تنفيذه. إن الدالة waitForCompletion تأخذ متحول وحيد يعبر عن السماح أو عدم السماح بالخرج المضجر، فعند السماح به سيتم طباعة عملية التقدم في تنفيذ العمل على شاشة الخرج. كما أن هذه الدالة تعود بقيمة منطقية تعبر عن نجاح العمل أو فشله.

3-2 التنفيذ التجريبي

بعد كتابة عمل المقابلة والاختزال، يكون من الطبيعي تجريبه على مجموعة بيانات تجريبية صغيرة من أجل حل أي مشاكل قد تظهر في الشيفرة البرمجية. بالتالي علينا تنصيب Hadoop بالنمط المستقل (standalone). حيث يتم في هذا النمط تنفيذ الأعمال محليا كما يتم الاعتماد على نظام الملفات المحلي.

وعند تنفيذ المثال السابق من أجل خمسة أسطر سنحصل على الخرج التالي (علما أنه تم حذف بعض الأسطر كما قمنا بالسماح بالتفاف النص لمناسبة العرض ضمن الصفحة).

```
% export HADOOP_CLASSPATH=Hadoop-examples.jar
% Hadoop MaxTemperature input/ncdc/sample.txt output
14/09/16 09:48:39 WARN util.NativeCodeLoader: Unable to load native-Hadoop
library for your platform... using builtin-java classes where applicable
14/09/16 09:48:40 WARN Map-Reduce .JobSubmitter: Hadoop command-line option
parsing not performed. Implement the Tool interface and execute your application
with ToolRunner to remedy this.
14/09/16 09:48:40 INFO input.FileInputFormat: Total input paths to process : 1
14/09/16 09:48:40 INFO Map-Reduce .JobSubmitter: number of splits:1
14/09/16 09:48:40 INFO Map-Reduce .JobSubmitter: Submitting tokens for job:
job_local26392882_0001
14/09/16 09:48:40 INFO Map-Reduce .Job: The url to track the job: http://localhost:8080/
14/09/16 09:48:40 INFO Map-Reduce .Job: Running job: job_local26392882_0001
14/09/16 09:48:40 INFO mapred.LocalJobRunner: OutputCommitter set in config null
14/09/16 09:48:40 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.Hadoop.Map-Reduce .lib.output.FileOutputCommitter
14/09/16 09:48:40 INFO mapred.LocalJobRunner: Waiting for map tasks
14/09/16 09:48:40 INFO mapred.LocalJobRunner: Starting task: attempt_local26392882_0001_m_000000_0
```

14/09/16 09:48:40 INFO mapred.Task: Using ResourceCalculatorProcessTree : null

14/09/16 09:48:40 INFO mapred.LocalJobRunner:

14/09/16 09:48:40 INFO mapred.Task: Task:attempt_local26392882_0001_m_000000_0

is done. And is in the process of committing

14/09/16 09:48:40 INFO mapred.LocalJobRunner: map

14/09/16 09:48:40 INFO mapred.Task: Task 'attempt_local26392882_0001_m_000000_0'

done.

14/09/16 09:48:40 INFO mapred.LocalJobRunner: Finishing task: attempt_local26392882_0001_m_000000_0

14/09/16 09:48:40 INFO mapred.LocalJobRunner: map task executor complete.

14/09/16 09:48:40 INFO mapred.LocalJobRunner: Waiting for reduce tasks

14/09/16 09:48:40 INFO mapred.LocalJobRunner: Starting task: attempt_local26392882_0001_r_000000_0

14/09/16 09:48:40 INFO mapred.Task: Using ResourceCalculatorProcessTree : null

14/09/16 09:48:40 INFO mapred.LocalJobRunner: 1 / 1 copied.

14/09/16 09:48:40 INFO mapred.Merger: Merging 1 sorted segments

14/09/16 09:48:40 INFO mapred.Merger: Down to the last merge-pass, with 1

segments left of total size: 50 bytes

14/09/16 09:48:40 INFO mapred.Merger: Merging 1 sorted segments

14/09/16 09:48:40 INFO mapred.Merger: Down to the last merge-pass, with 1

segments left of total size: 50 bytes
14/09/16 09:48:40 INFO mapred.LocalJobRunner: 1 / 1 copied.
14/09/16 09:48:40 INFO mapred.Task: Task:attempt_local26392882_0001_r_000000_0
is done. And is in the process of committing
14/09/16 09:48:40 INFO mapred.LocalJobRunner: 1 / 1 copied.
14/09/16 09:48:40 INFO mapred.Task: Task attempt_local26392882_0001_r_000000_0
is allowed to commit now
14/09/16 09:48:40 INFO output.FileOutputCommitter: Saved output of task 'attempt...local26392882_0001_r_000000_0' to file:/Users/tom/book-workspace/
Hadoop-book/output/_temporary/0/task_local26392882_0001_r_000000
14/09/16 09:48:40 INFO mapred.LocalJobRunner: reduce > reduce
14/09/16 09:48:40 INFO mapred.Task: Task 'attempt_local26392882_0001_r_000000_0'
done.
14/09/16 09:48:40 INFO mapred.LocalJobRunner: Finishing task: attempt_local26392882_0001_r_000000_0
14/09/16 09:48:40 INFO mapred.LocalJobRunner: reduce task executor complete.
14/09/16 09:48:41 INFO Map-Reduce .Job: Job job_local26392882_0001 running in uber mode : false
14/09/16 09:48:41 INFO Map-Reduce .Job: map 100% reduce 100%
14/09/16 09:48:41 INFO Map-Reduce .Job: Job job_local26392882_0001 completed

```
successfully
14/09/16 09:48:41 INFO Map-Reduce .Job: Counters: 30
File System Counters
FILE: Number of bytes read=377168
FILE: Number of bytes written=828464
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
Map-Reduce Framework
Map input records=5
Map output records=5
Map output bytes=45
Map output materialized bytes=61
Input split bytes=129
Combine input records=0
Combine output records=0
Reduce input groups=2
Reduce shuffle bytes=61
Reduce input records=5
Reduce output records=2
Spilled Records=10
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=39
Total committed heap usage (bytes)=226754560
File Input Format Counters
```

Bytes Read=529

File Output Format Counters

Bytes Written=29

عند استدعاء الأمر Hadoop بحيث يكون أول متحول هو اسم الصف. يقوم Hadoop بتشغيل آلة جافا افتراضية من أجل تنفيذ الصف (الخاص بالعمل). يقوم الأمر Hadoop بإضافة مكاتب Hadoop إلى مسار الصفوف classpath كما يقوم بتحميل الإعدادات الخاصة بـHadoop .

إن الخرج الناتج عن تنفيذ العمل السابقة يحتوي على كمية مهمة من المعلومات. على سبيل المثال يمكننا أن نرى أن العمل قد حصل على معرف هو job_local26392882_0001 كما أنه قام بتشغيل مهمة مقابلة واحدة ومهمة اختزال واحدة ولكل مهمة معرف

attempt_local26392882_0001_m_000000_0,)

(attempt_local26392882_0001_r_000000_0

إن المعرف الخاص بالعمل والمعرفات الخاصة بمهام المقابلة والاختزال تساعد كثيرا عند عملية التنقيح.

إن القسم الأخير من الخرج والمعنون "Counters" يظهر الإحصائيات التي يولدها Hadoop والخاصة بالعمل، إن هذه الإحصائيات تفيد كثيرا للتأكد من أن كمية البيانات التي تمت معالجتها هي نفسها الكمية المتوقعة. على سبيل المثال، يمكننا أن نتفحص عدد السجلات التي تدفقت عبر النظام: حيث لدينا خمسة سطور دخل للمقابل تنتج خمس سجلات خرج (لأن المقابل يقوم بإرسال سجل خرج واحد من أجل كل سجل دخل صحيح)، ولدينا خمسة سجلات دخل خاصة بالاختزال (ولكل سجل مفتاح فريد) ونلاحظ خرج المختزل هو عبارة عن سجل واحد.

تمت كتابة الخرج إلى المجلد ouput_directory، والذي يحتوي على ملف خرج واحد من أجل كل مختزل. وبما أن العمل الذي نقوم به في مثالنا يحتوي على مختزل واحد بالتالي سنجد ملف خرج واحد ويدعى *part–r–*00000 وسيحتوي هذا الملف على النتيجة التالية.

```
% cat output/part-r-00000
1949 111
1950 22
```

إن هذه النتيجة هي نفسها النتيجة التي كنا سنحصل عليها لو قمنا بالحساب بشكل يدوي. وهذه النتيجة تعني أن درجة الحرارة الأعظمية لعام 1949 هي 11.1 درجة مئوية ومن أجل عام 1950 هي 2.2 درجة مئوية.

2–4 التوسع

المثال السابق يتعامل مع بيانات صغيرة مخزنة على نظام الملفات المحلي. من أجل توسعة العمل، نحن بحاجة لتخزين البيانات على نظام ملفات موزع (عادة هو نظام الملفات الموزع الخاص ب HDFS Hadoop). إن ذلك يسمح لHadoop بنقل حساب المقابلة والاختزال إلى أي عقدة تحتوي على جزء من البيانات وذلك من خلال مدير الموارد الذي يدعى YARN.

2–5 تحديد تابع التجميع

بالعودة إلى برنامج المقابلة والاختزال المكتوب بلغة java، يتم تحديد تابع التجميع باستخدام الصف Reducer، وبالنسبة لمثالنا فإن تابع التجميع هو نفسه تابع الاختزال. والتعديل الوحيد يكون بتحديد تابع التجميع من أجل العمل.

```
public class MaxTemperatureWithCombiner {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperatureWithCombiner <input path> " +
          "<output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperatureWithCombiner.class);
    job.setJobName("Max temperature");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setCombinerClass(MaxTemperatureReducer.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
```

```
        job.setOutputValueClass(IntWritable.class);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

الشكل (15) تحديد تابع التجميع من أجل عمل المقابلة والاختزال في Java

المراجع

[1] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Proc. 6th Symp. Oper. Syst. Des. Implement.*, pp. 137–149, 2004, doi: 10.1145/1327452.1327492.

[2] T. White, *Hadoop: The definitive guide*, vol. 54. 2015.

[3] K. Seki, R. Jinno, and K. Uehara, “Parallel Distributed Trajectory Pattern Mining Using Hierarchical Grid with MapReduce,” vol. 5, no. December, pp. 79–96, 2013, doi: 10.4018/ijghpc.2013100106.

[4] “SETI@home.” https://setiathome.berkeley.edu/ (accessed Jul. 02, 2021).

[5] “HDFS design.” https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (accessed Jul. 02, 2021).

[6] “YARN.” https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html (accessed Jul. 02, 2021).

[7] “Spark,” [Online]. Available: https://spark.apache.org/.

[8] “Impala.” https://impala.apache.org/ (accessed Jul. 02, 2021).

[9] “Hadoop: Capacity Scheduler.” https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html (accessed Jul. 02, 2021).

[10] “Hadoop: Fair Scheduler.” https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html (accessed Jul. 02, 2021).

[11] “MapReduce Tutorial.” https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html (accessed Jul. 02, 2021).

[12] K. Wang, G., Butt, A. R., Pandey, P., & Gupta, “A simulation approach to evaluating design decisions in mapreduce setups,” *Model. Anal. Simul.*

Comput. Telecommun. Syst. 2009. MASCOTS'09. IEEE Int. Symp., 2009.

[13] "Mumak: Map-Reduce Simulator." https://issues.apache.org/jira/browse/MAPREDUCE-728 (accessed Jun. 11, 2018).

[14] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, and K. T. A. Chien, "The MicroGrid : a Scientific Tool for Modeling Computational Grids," vol. 00, no. c, pp. 1–22, 2000.

[15] "SimGrid." http://simgrid.gforge.inria.fr/ (accessed Jul. 02, 2021).

[16] "GridSim." https://sourceforge.net/projects/gridsim/ (accessed Jul. 02, 2021).

[17] "SimJava." http://www.dcs.ed.ac.uk/home/hase/simjava/ (accessed Jul. 02, 2021).

[18] S. Hammoud, Maozhen Li, Yang Liu, Nasullah Khalid Alham, and Zelong Liu, "MRSim: A discrete event based MapReduce simulator," *2010 Seventh Int. Conf. Fuzzy Syst. Knowl. Discov.*, no. Fskd, pp. 2993–2997, 2010, doi: 10.1109/FSKD.2010.5569086.

[19] X. Zhou, W. Luo, and H. Tan, "MRTune: A simulator for performance tuning of MapReduce jobs with skewed data," *Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS*, vol. 2015-April, pp. 352–359, 2014, doi: 10.1109/PADSW.2014.7097828.

[20] WikiPedia, "zipf," 2021. https://en.wikipedia.org/wiki/Zipf%27s_law (accessed Jul. 02, 2021).

[21] Y. Liu, M. Li, N. Khalid, and S. Hammoud, "HSim : A MapReduce simulator in enabling Cloud Computing," *Futur. Gener. Comput. Syst.*, vol. 29, no. 1, pp. 300–308, 2013, doi: 10.1016/j.future.2011.05.007.

[22] M. Bateman and S. Bhatti, "TCP testing : How well does ns2 match reality ?," *2010 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, pp. 276–284, 2010, doi: 10.1109/AINA.2010.133.

[23] N. Liu, X. Yang, X. H. Sun, J. Jenkins, and R. Ross, "YARNsim: Simulating

hadoop YARN," *Proc. - 2015 IEEE/ACM 15th Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2015*, pp. 637–646, 2015, doi: 10.1109/CCGrid.2015.61.

[24] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS : A High-Performance , Low Memory , Modular Time Warp System," 2000, doi: 10.1109/PADS.2000.847144.

[25] "CODES." http://press3.mcs.anl.gov/codes/ (accessed Jul. 02, 2021).

[26] W. Kolberg, P. De B. Marcos, J. C. S. Anjos, A. K. S. Miyazaki, C. R. Geyer, and L. B. Arantes, "MRSG - A MapReduce simulator over SimGrid," *Parallel Comput.*, vol. 39, no. 4–5, pp. 233–244, 2013, doi: 10.1016/j.parco.2013.02.001.

[27] R. Singhal and A. Verma, "Predicting job completion time in heterogeneous MapReduce environments," *Proc. - 2016 IEEE 30th Int. Parallel Distrib. Process. Symp. IPDPS 2016*, pp. 17–27, 2016, doi: 10.1109/IPDPSW.2016.10.

[28] Z. Ren, Z. Liu, X. Xu, J. Wan, W. Shi, and M. Zhou, "Waxelephant: A realistic hadoop simulator for parameters tuning and scalability analysis," *Proc. - 7th ChinaGrid Annu. Conf. ChinaGrid 2012*, pp. 9–16, 2012, doi: 10.1109/ChinaGrid.2012.25.

[29] S. Ning, F. T. B, Y. Li, Z. Cui, and L. Yu, *RUPredHadoop: Resources Utilization Predictor for Hadoop with Large-Scale Clusters*, vol. 945, no. 2017. Springer Singapore, 2018.

[30] "Hadoop." https://hadoop.apache.org/ (accessed Jul. 02, 2021).

[31] A. Legrand, "Scheduling for Large Scale Distributed Computing Systems: Approaches and Performance Evaluation Issues," 2015.

[32] "NS2." http://nsnam.sourceforge.net/wiki/index.php/Main_Page (accessed Jul. 02, 2021).

[33] "omnet++." https://omnetpp.org/ (accessed Jul. 02, 2021).

[34] "opnet." http://www.opnet.com.tw/ (accessed Jul. 02, 2021).

[35] T. Hoefler, T. Schneider, and A. Lumsdaine, “LogGOPSim,” p. 597, 2010, doi: 10.1145/1851476.1851564.

[36] F. Ino, N. Fujimoto, and K. Hagihara, “LogGPS: A parallel computational model for synchronization analysis,” *SIGPLAN Not. (ACM Spec. Interes. Gr. Program. Lang.*, vol. 36, no. 7, pp. 133–142, 2001, doi: 10.1145/568014.379592.

[37] “SimGrid Model.” https://simgrid.org/tutorials/surf-101.pdf (accessed Jul. 02, 2021).

[38] M. Quinson *et al.*, “Parallel Simulation of Peer-to-Peer Systems To cite this version : HAL Id : inria-00602216 Parallel Simulation of Peer-to-Peer Systems,” 2013.

[39] H. Casanova, A. Legrand, and M. Quinson, “SimGrid: A generic framework for large-scale distributed experiments,” in *Proceedings - UKSim 10th International Conference on Computer Modelling and Simulation, EUROSIM/UKSim2008*, 2008, pp. 126–131, doi: 10.1109/UKSIM.2008.28.

[40] A. Lebre, A. Legrand, F. Suter, and P. Veyre, “Adding storage simulation capacities to the SimGrid toolkit: Concepts, models, and API,” *Proc. - 2015 IEEE/ACM 15th Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2015*, pp. 251–260, 2015, doi: 10.1109/CCGrid.2015.134.

[41] “hdd slice.” https://www.kernel.org/doc/Documentation/block/cfq-iosched.txt (accessed Jul. 02, 2021).

[42] “oeis.” https://oeis.org/A090657 (accessed Jul. 02, 2021).

[43] “the total size of all the image sets of all functions from [n] to [n],” [Online]. Available: https://oeis.org/A152170.

[44] F. Tian, T. Ma, B. Dong, and Q. Zheng, “PWLM3-based automatic performance model estimation method for HDFS write and read operations,” *Futur. Gener. Comput. Syst.*, vol. 50, pp. 127–139, 2015, doi: 10.1016/j.future.2015.01.011.

[45] “seagate 15k.” https://www.seagate.com/files/docs/pdf/datasheet/disc/ds-

cheetah-15k-6-us.pdf (accessed Aug. 20, 2021).

[46] "seagate 18k." https://www.seagate.com/www-content/datasheets/pdfs/3-5-barracudaDS1900-11-1806US-en_US.pdf (accessed Aug. 20, 2021).

[47] "Pig," [Online]. Available: https://pig.apache.org/.

[48] "Hive." https://hive.apache.org/ (accessed Jul. 02, 2021).

[49] "Hbase," [Online]. Available: https://hbase.apache.org/.

[50] L. and D. W. Kelton, *Simulation modelling and analysis*, vol. 19, no. 3. 1999.

[51] "Zookeeper." https://zookeeper.apache.org/ (accessed Jul. 02, 2021).

[52] S. Dhanalakshmi, B. Arputhamary, and L. Arockiam, "A Survey on Data Skew Mitigating Techniques in Hadoop MapReduce Framework," vol. 5, no. 7, pp. 290–296, 2016.

Syrian Arab Republic

Ministry of Higher Education

Al-Baath University

Faculty of Informatics Engineering

Department of System Engineering and Computer Networks

Simulating Map-Reduce Framework

A thesis submitted in fulfillment of the requirements for the PHD Degree in System Engineering and Computer Networks

Prepared by:

Eng. Amer Ahmad Dwar

Supervised by:

Prof. Muhammad Al-Nayef Al-Haj Younis

PhD. Suhail Al-Hammoud

1442 A.H – 2021 A.D