آناتومی کارهای نگاشت‌کاهش و بررسی چالش‌های زمان‌بندی، کارایی و امنیت در هادوپ

محمدعلی ترکمانی

گروه مهندسی کامپیوتر، واحد یاسوج، دانشگاه آزاد اسلامی، یاسوج ، ایران، torkamani_ali@yahoo.com

واحد آموزش و پژوهش، کارخانجات مخابراتی ایران، شیراز، ایران، Torkamani@itmc.ir

کیوان رحیمی زاده

استادیار، دانشکده فنی و مهندسی، دانشگاه یاسوج، یاسوج، ایران، Rahimizadeh@yu.ac.ir

چکیده

هادوپ یک ابزار متن باز مبتنی بر رهیافت نگاشت-کاهش در زمینه‌ی ذخیره‌سازی و پردازش داده‌های کلان است که تحقق انقلاب داده های کلان را بیش‌تر از پیش ممکن ساخته است. به جای تکیه بر سخت‌افزارهای اختصاصی گران‌قیمت، خوشه‌های هادوپ معمولاً از صدها هزار ماشین معمولی چندهسته‌ای تشکیل شده‌اند. هادوپ کدها را به جای انتقال داده‌ها به گره‌های پردازشی، به ماشینی که داده‌ها در آنها قرار دارند منتقل می‌کند که این امر مقیاس‌پذیری را افزایش می‌دهد. قابلیت پردازش مقادیر بالای داده‌های متنوع به شکل توزیع‌شده و موازی همراه با تحمل‌پذیری خطا و استفاده از نرم‌افزارهای رایگان و سخت‌افزارهای مناسب ارزان‌قیمت، استفاده از هادوپ را به عنوان سکوی انتخابی داده های کلان برای بیشتر سازمان‌های تجاری و دولتی توجیه‌پذیر می‌کند. با این حال، ساخت یک کار نگاشت‌کاهش که چند ترابایت داده را در بین صدها ماشین در زمانی معقول خوانده و پردازش کند، چالش بزرگی است. علاوه بر این، روش‌های حفاظتی سنتی داده‌ها برای داده‌های کلان قابل استفاده نیستند، به همان اندازه که حجم داده‌ها افزایش می‌یابد، ریسک‌های امنیتی نیز شدیدتر می‌شود. سکوهای داده‌های کلان به طور خاص برای پشتیبانی از فرم پیشرفته‌ای از تحلیل طراحی شده‌اند که کارایی دقیق و الزامات مقیاس‌پذیری را برآورده می‌کند. با این حال، تاکنون توجه خاصی معطوف حفاظت از داده‌ها نشده است. در واقع، با وجود آنکه داده‌های تحلیل‌شده اغلب حاوی اطلاعات شخصی و حساس هستند و تهدیداتی در ارتباط با امنیت و حریم خصوصی متوجه آنها است. این مقاله ابتدا اکوسیستم هادوپ را توصیف کرده و جزییاتی را در مورد موتور نگاشت‌کاهش موجود در هسته هادوپ ارائه می‌کند. همچنین یک برنامه ساده نگاشت کاهش به زبان جاوا بررسی خواهد شد که این برنامه میتواند رشته‌ها را معکوس کند. سپس به بحث در مورد برنامه‌های زمان‌بندی نگاشت‌کاهش مختلف و تأثیر آن‌ها بر کارایی می‌پردازد. همچنین چالشهای امنیتی هادوپپ بررسی خواهد شد.

کلیدواژه‌ها: داده‌های کلان، فایل‌سیستم توزیع‌شده هادوپ، کارایی، زمان‌بندی، امنیت، نگاشت\کاهش هادوپ.

1- مقدمه

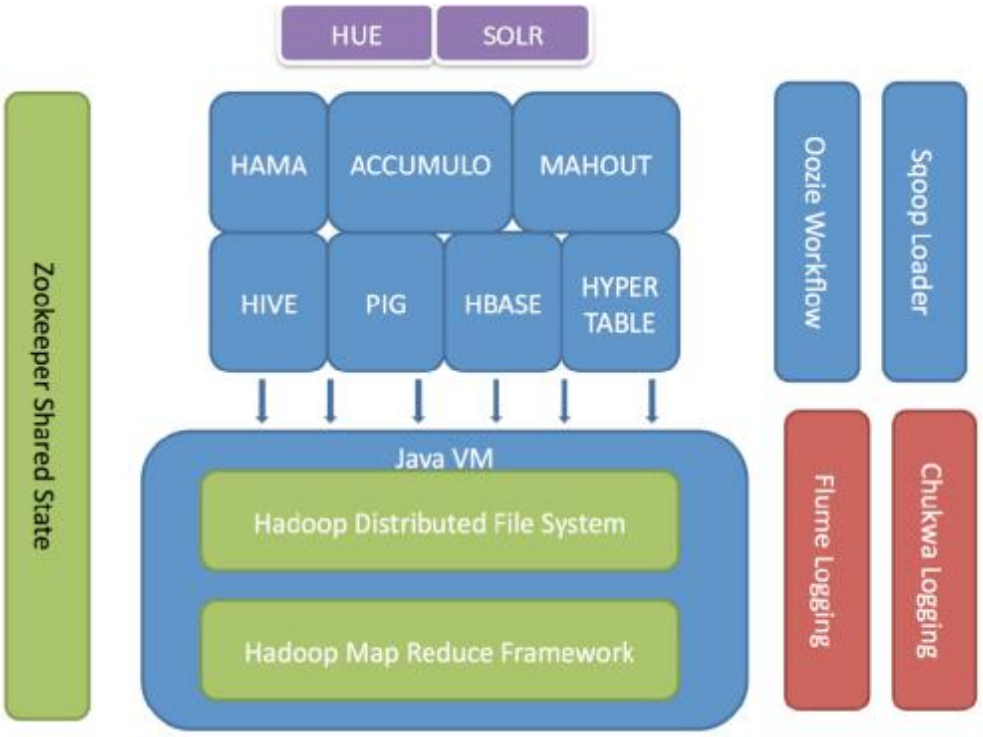
«آپاچی هادوپ»[1]، به شکلی که در حال حاضر ارائه می‌شود، دارای نزدیک به نیم میلیون خط کد جاوا و صدها پارامتر قابل تنظیم است [1]. تنظیم پارامترهای هادوپ برای کارایی مطلوب انواع برنامه‌های کاربردی، هدفی دست‌نیافتنی است. اما، درکی عمیق از فازهای مختلف وظیفه «نگاشت‌کاهش»[2]، این امکان را برای توسعه‌دهندگان و کارکنان عملیات‌ها فراهم می‌آورد تا یک خوشه هادوپ را بهتر تنظیم کرده و به کارایی نزدیک به سطح مورد نظر خود دست پیدا کنند. هادوپ دارای دو جز اصلی است: «سیستم فایل توزیع‌شده»[3] (DFS) و چارچوب نگاشت‌کاهش. نگاشت‌کاهش قادر به حل طیف گسترده‌ای از مسائل محاسباتی است. هادوپ شالوده‌ای مشترک را برای چند سیستم مانند «آپاچی پیگ»[4] [2]، «آپاچی هایو»[5] [3] و «آپاچی اکومولو»[6] [4] فراهم می‌آورد تا این سیستم‌ها مجبور به ساخت ذخیره‌سازی پردازش فایل پیچیده و توزیع‌شده خود و روش‌های بازیابی فایل همراه با آن نباشند. هادوپ تا حد زیادی مبتنی بر نگاشت‌کاهش بوده و بنابراین درک فرآیندهای داخلی نگاشت‌کاهش از اهمیت ویژه‌ای برخوردار است. تنظیم یک سیستم نگاشت‌کاهش به دلیل تعداد بالای پارامترهای تنظیمی، کار بسیار دشواری است. همچنین، معماری کنونی هادوپ دارای مشکلات کارایی بالقوه برای خوشه‌های بسیار بزرگ است چون دارای معماری یک ردیاب وظیفه به ازای هر وظیفه است. جامعه هادوپ در حال تلاش برای حل این مسائل هستند و ما در این مقاله به بحث در مورد مهم‌ترین آنها خواهیم پرداخت. بقیه مثاله به صورت زیر سازماندهی شده است. بخش 2 اکوسیستم هادوپ را شرح داده و اجزای نگاشت‌کاهش را از طریق یک مثال معرفی می‌کند. بخش 3 نگاهی جزئی‌تر به فازهای مختلف یک وظیفه نگاشت‌کاهش می‌اندازد. بخش 4 به نوشتن یک برنامه از نگاشت کاهش هادوپ اختصاص دارد. بخش 5 به بحث در مورد چالش‌های زمان‌بندی یک خوشه نگاشت‌کاهش می‌پردازد. بخش 6 چالش‌های کارایی نگاشت‌کاهش و بخش 7 چالش‌ها و مشکلات امنیتی هادوپ را بررسی می‌کند.
بخش نهایی مقاله نیز نتیجه‌گیری ارائه میگردد.

2- اکوسیستم هادوپ و نگاشت‌کاهش

فهرست بلندی از محصولات و پروژه‌هایی وجود دارد که کاربردهای هادوپ را گسترش داده یا از یکی از قابلیت‌های موجود هادوپ به شیوه‌ای جدید بهره می‌برند. به عنوان مثال، اجرای پرسمان‌های SQL-مانند بر روی هادوپ منجر به به وجود آمدن چندین محصول شده است. این حرکت را فیسبوک با به وجود آوردن HiveQL آغاز کرد. اکومولو [4] و HBase [5]، هر دو پایگاه‌های داده‌ای NoSQL (یعنی ذخیره‌سازی جدولی نیمه-ساختاریافته) هستند که بر روی هادوپ ساخته شده‌اند. نوشته شدن وظایف نگاشت‌کاهشی به دست کاربران که داده‌ها را از یک منبع ذخیره‌سازی NoSQL مانند HBASE یا اکومولو فراخوانی کرده و داده‌ها را بر روی آن‌ها می‌نویسند، امری رایج است. در نهایت، یک پروژه آپاچی با نام Pig [2] یک لایه «انتزاع» را با کمک اسکریپت‌هایی در زبان Pig Latin که به وظایف نگاشت‌کاهش ترجمه می‌شوند، ارائه می‌کند. دیگر نمونه‌های پروژه‌های ساخته‌شده بر روی هادوپ عبارتند از Apache Sqoop [6]، Apache

1 Apache Hadoop
2 MapReduce
3 distributed file system
4 Apache Pig
5 Apache Hive
6 Apache Accumulo

Oozie [7] و Apache Flume [8]. شکل 1 برخی از محصولاتی را نشان می‌دهد که بر روی هادوپ ساخته شده و آن را کامل می‌کنند.



شکل 1. اکوسیستم هادوپ با محصولات بسیار.

3- آناتومی نگاشت‌کاهش

جزئیات درون نگاشت‌کاهش در این بخش از چند منبع مختلف مانند [9، 10، 11، 12، 13] نشأت می‌گیرند. مدل برنامه‌نویسی نگاشت‌کاهش از یک تابع نگاشت <k1; v1> و یک تابع کاهش <k2; list(v2)> تشکیل شده است. کاربران با مشخص کردن یک تابع سفارشی map و reduce که با یک زبان برنامه‌نویسی عمومی مانند جاوا یا پایتون نوشته شده باشد، می‌توانند منطق پردازشی خود را پیاده‌سازی کنند. تابع نگاشت < k1; v1> به ازای هر جفت مقدار کلیدی <k1; v1> در داده‌های ورودی فراخوانی شده تا به خروجی صفر یا جفت‌های مقدار کلیدی بیش‌تر با شکل <k2; v2> برسد. تابع کاهش <k2; list(v2)> برای هر کلید منحصر به فرد k2 و مقادیر منطبق list(v2) از خروجی نگاشت، فراخوانی می‌شود. خروجی تابع کاهش <k2; list(v2)> صفر یا جفت‌های مقادیر کلیدی بیشتر با شکل <k3; v3> است. مدل برنامه‌نویسی نگاشت‌کاهش همچنین به دیگر توابع مانند partition(k2) اجازه می‌دهد تا نحوه تقسیم شدن جفت‌های مقادیری کلیدی خروجی نگاشت در میان وظایف کاهش را کنترل کند و به تابع ترکیب <k2; list(v2)> اجازه می‌دهد تا تجمع جزیی را در سمت نگاشت اجرا کند. کلیدهای k1، k2 و k3 و مقادیر v1، v2 و v3 می‌توانند دارای انواع مختلفی باشند. شکل 2 جریان داده‌ها از فازهای نگاشت و کاهش را نشان می‌دهد. یک خوشه‌ی نگاشت‌کاهش هادوپ از یک معماری سرویس گیرنده-سرویس دهنده استفاده می‌کند که در آن یک

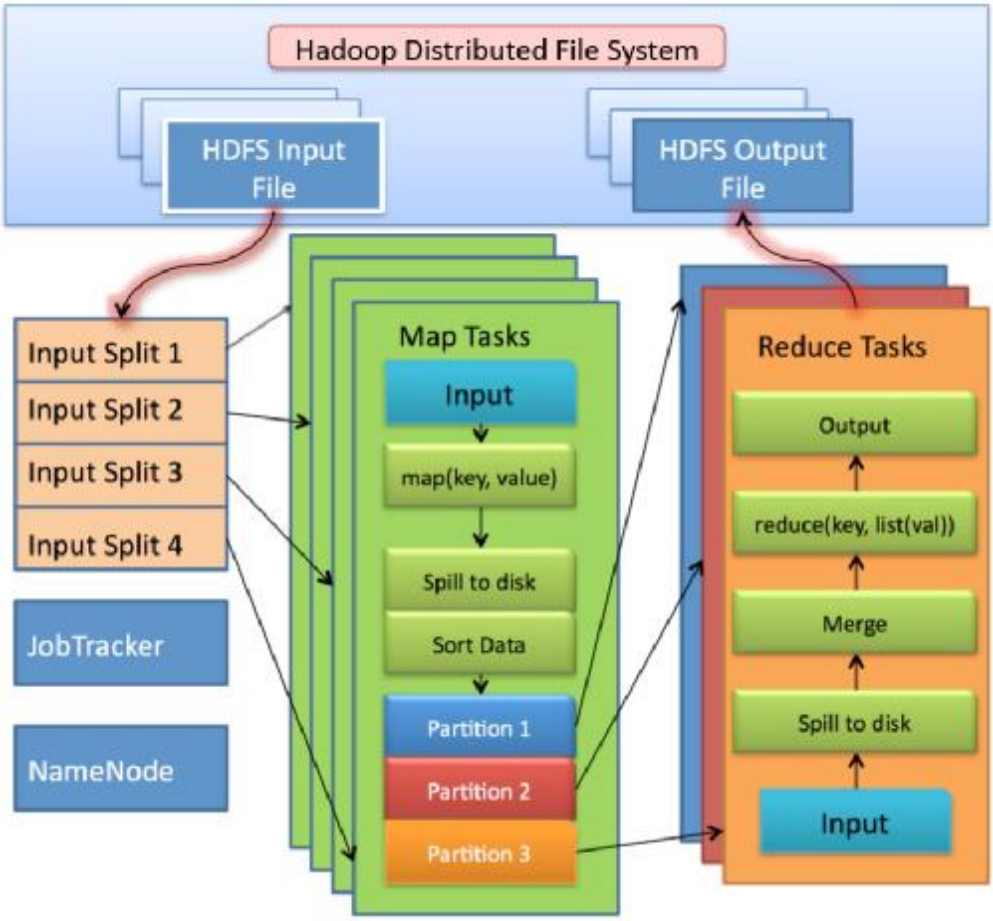
گره سرویس گیرنده (که با نام «ردیاب کار»[1] شناخته می‌شود) چند گره کارگر (که با نام «ردیاب وظیفه»[2] شناخته می‌شوند) را مدیریت می‌کند. هادوپ یک وظیفه نگاشت‌کاهش را ابتدا با تقسیم‌بندی منطقی مجموعه داده‌ها به چند بخش داده‌ای اجرا می‌کند. سپس هر وظیفه نگاشت به یک گره ردیاب وظیفه که آن بخش داده‌ای در آن قرار داد سپرده می‌شود. یک برنامه زمان‌بندی وظایف مسئول زمان‌بندی اجرای وظایف تا جای ممکن در یک حالت داده‌های محلی است. چند نوع برنامه زمان‌بندی مختلف تا به حال برای محیط نگاشت‌کاهش طراحی شده‌اند. از دید بالا به پایین، یک وظیفه نگاشت‌کاهش آنقدرها هم پیچیده نیست چون هادوپ بیشتر پیچیدگی‌های نوشتن برنامه‌های موازی برای یک خوشه از رایانه‌ها را پنهان می‌کند. در یک خوشه هادوپ، هر گره معمولاً چند وظیفه‌ی نگاشت را (چندین بار بسته به تعداد هسته‌های یک ماشین) آغاز می‌کند و هر وظیفه قسمتی از داده‌های ورودی را در یک توالی می‌خواند، تمام ردیف‌های داده را پردازش کرده و یک جفت <key, value> را به عنوان خروجی ارائه می‌کند. وظایف کاهش‌دهنده هم در ادامه کلیدها و مقادیر خروجی وظایف نگاشت‌کننده را جمع‌آوری کرده و کلیدهای یکسان را به یک کلید و مقادیر نگاشت مختلف را به مجموعه‌ای از مقادیر تبدیل می‌کنند. سپس یک کاهش‌دهنده واحد بر روی این کلیدهای ادغام‌شده کار کرده و وظیفه کاهش‌دهنده داده‌های مورد نظر خود را با بررسی ورودی‌های کلیدهای خود و مجموعه مقادیر مرتبط به آن، به عنوان خروجی تولید می‌کند. برنامه‌نویس تنها باید یک منطق و کد را برای توابع map و reduce فراهم کند. این پارادایم ساده می‌تواند بسیاری از مسائل محاسباتی را حل کند و یکی از ستون‌های انقلاب پردازش کلان‌داده‌ها به شمار می‌رود. در یک وظیفه‌ی نگاشت‌کاهش معمولی، فایل‌های ورودی از «سیستم فایل توزیع‌شده‌ی هادوپ» (HDFS) خوانده می‌شوند. داده‌ها معمولاً فشرده می‌شوند تا حجم فایل‌ها کاهش پیدا کند. پس از خارج شدن از حالت فشرده، بایت‌های سریالی پیش از انتقال به تابع map تعریف شده توسط کاربر، تبدیل به اشیای جاوا می‌شوند. در مقابل، رکوردهای خروجی سریالی شده، فشرده‌سازی می‌شوند و در نهایت به HDFS باز می‌گردند. با این حال، در پشت این سادگی ظاهری، این پردازش به چندین مرحله تقسیم شده و دارای صدها پارامتر قابل تنظیم مختلف برای تنظیم کردن دقیق ویژگی‌های در حال اجرای وظیفه است. ما این مراحل را با جزییات شرح خواهیم داد، از زمانی که یک وظیفه آغاز می‌شود تا جایی که تمام وظایف نگاشت و کاهش تکمیل شده‌اند و «ردیاب کار» (JT) کار را پاک می‌کند.

3-1- فاز راه‌اندازی

«ارایه کار» نقطه‌ی آغازین چرخه زندگی یک کار نگاشت‌کاهش است. یک کار هادوپ بر روی یک ماشین واحد ارائه شده و این کار باید نسبت به آدرس‌های ماشین‌هایی که در آنها دیمن‌های «گره نام»[3] (یک دیمن سرویس گیرنده واحد در خوشه‌ی هادوپ) و ردیاب کار در حال اجرا هستند، آگاه باشد. چارچوب در ابتدا تمام منابعی (مانند فایل‌های jar و تنظیمات جاوا) را که باید در HDFS توزیع شوند ذخیره می‌کند. اینها شامل منابعی می‌شوند که از طریق پارامترهای مختلف بر روی استدلال‌های خط فرمان ارائه شده‌اند، به علاوه فایل JAR که به عنوان فایل JAR کار مشخص شده است. این مرحله به شکل یک توالی بر روی ماشین محلی اجرا می‌شود. نسخه XML داده‌های تنظیمات کار هم در HDFS ذخیره می‌شود. سپس چارچوب مجموعه

1 JobTracker
2 TaskTracker
3 NameNode

داده‌های ورودی را بررسی کرده و با استفاده از کلاس InputFormat مشخص‌شده در تنظیمات کار تعیین می‌کند که کدام فایل‌های ورودی باید به طور کامل به یک وظیفه انتقال داده شده و کدام فایل‌های ورودی باید در میان چند وظیفه تقسیم شوند. به منظور تعیین نحوه‌ی اجرای وظایف نگاشت، این چارچوب از پارامترهای مختلفی استفاده می‌کند. جزییات InputFormat می‌توانند پارامتری مانند mapred.map.tasks$ (که یک پارامتر کار برای نشان دادن تعداد وظایف نگاشت مورد نیاز کاربر است) را باطل کنند. به عنوان مثال، یک فرمت ورودی خاص می‌تواند تقسیم‌بندی‌ها را بر اساس تعداد خط‌ها تحمیل کند.



شکل 2. جریان داده‌ی نگاشت‌کاهش هادوپ.

در یک خوشه هادوپ، در بسیاری از مواقع تعداد اسلات‌های موجود برای وظایف با تعداد هسته‌های CPU در دسترس بر روی گره‌های کارگر مرتبط است. اندازه تقسیم ایده‌آل برابر است با اندازه یک یا چند بلوک سیستمِ فایل، چون این اندازه به چارچوب اجازه می‌دهد تا سعی کند داده‌ها را به طور محلی برای وظیفه‌ای که تقسیم را پردازش می‌کند فراهم کند. نتیجه این فرآیند مجموعه‌ای از تقسیم‌های ورودی است که هرکدام با اطلاعات مربوط به این که کدام ماشین‌ها دارای کپی‌های محلی داده‌های تقسیم‌شده هستند، برچسب خورده‌اند. تقسیم‌ها بر اساس اندازه مرتب می‌شوند تا بزرگترین تقسیم‌ها زودتر اجرا شوند. اطلاعات تقسیم و اطلاعات تنظیمات کار برای اجرا از طریق یک فایل اطلاعات کار که در HDFS ذخیره شده است به ردیاب

کار ارسال می‌شوند. خطوط اسنیپت یک کلاس اصلی کار نگاشت‌کاهش معمول («شمارنده‌ی کلمات»[1] به عنوان یک مثال) در زیر آورده شده‌اند. کار و کلاس‌های همراهان آن مانند کار-کلاینت مسئول ارائه یک کار نگاشت‌کاهش هستند. پس از ارائه کار، متد waitForCompletion پیشرفت کار را هر یک ثانیه بررسی کرده و اگر تغییری در پیشرفت کار به وجود آمده باشد آن را به کنسول گزارش می‌کند. پس از اتمام کار، اگر موفقیت‌آمیز بوده باشد، شمارنده کارها نمایش داده می‌شود. در غیر این صورت، خطایی که منجر به شکست کار شده است در کنسول ذخیره می‌شود. فرآیند ارائه کارِ استفاده‌شده توسط متد submitJobInternal کار-کلاینت، به طور خلاصه این کارها را انجام می‌دهد:

- از ردیاب کار درخواست ID یک کار جدید را می‌کند (با فراخوانی getNewJobId بر روی ردیاب کار).
- مشخصات خروجی کار را بررسی می‌کند. به عنوان مثال، اگر دایرکتوری خروجی مشخص نشده باشد یا از پیش وجود داشته باشد، یک خطا نمایش داده شده و ارائه کار متوقف می‌شود.
- تقسیم‌های ورودی برای کار را محاسبه می‌کند. در صورت وقوع خطا، کار ارائه نشده و یک خطا در برنامه نگاشت‌کاهش نمایش داده می‌شود.
- منابع مورد نیاز برای اجرای کار، از جمله فایل JAR کار، فایل تنظیمات و تقسیم‌های ورودی محاسبه‌شده، را به فایل‌سیستم ردیاب کار در یک دایرکتوری هم‌نام ID کار کپی می‌کند. فایل JAR کار با یک ضریب تکرار بالا (که توسط ویژگی mapred.submit.replication$ کنترل می‌شود و به طور پیش‌فرض برابر با 10 است) کپی می‌شود تا کپی‌ها به تعداد کافی در سرتاسر خوشه برای دسترسی ردیاب‌های کار در هنگام اجرای وظایف برای کار وجود داشته باشند.
- به ردیاب کار اطلاع می‌دهد که کار آماده اجرا است (با فراخوانی submitJob بر روی ردیاب کار).

زمانی که ردیاب کار یک فراخوانی را در متد submitJob خود دریافت می‌کند، آن را در یک صف داخلی قرار می‌دهد تا برنامه زمان‌بندی کار بتواند آن را از آنجا برداشته و آغاز کند. آغاز کار شامل ایجاد یک شیء به نمایندگی از کار در حال اجرا، که وظایف‌اش را در بر می‌گیرد و ثبت اطلاعات به منظور ردیابی وضعیت کار و پیشرفت آن است. به منظور ایجاد فهرست وظایف برای اجرا، برنامه زمان‌بندی کار ابتدا تقسیم‌های ورودی محاسبه‌شده توسط کار-کلاینت را از فایل‌سیستم مشترک دریافت می‌کند. سپس این برنامه یک وظیفه نگاشت برای هر تقسیم ایجاد می‌کند. تعداد وظایف کاهشی که باید ایجاد شوند به وسیله ویژگی mapred.reduce.tasks$ در شیء JobConf تعیین شده، که توسط متد setNumReduceTasks مشخص می‌شود، و برنامه زمان‌بندی تنها این تعداد از وظایف کاهش برای اجرا را ایجاد می‌کند. در این زمان برای وظایف ID تعیین می‌شود. ردیاب‌های وظیفه یک حلقه ساده را اجرا می‌کنند که فراخوان‌های متد ضربان قلبی را به صورت دوره‌ای به ردیاب کار ارسال می‌کند. تعداد ردیاب‌های وظیفه در یک خوشه هادوپ برابر است با تعداد گره‌های کارگر (که می‌توانند صدها یا هزاران گره باشند). ضربان‌ها به ردیاب کار اطلاع می‌دهند که یک ردیاب وظیفه زنده است، اما به عنوان حامل پیام هم انجام وظیفه می‌کنند. به عنوان بخشی از ضربان، ردیاب وظیفه مشخص می‌کند که آیا آمادگی لازم برای اجرای یک وظیفه جدید را دارد یا خیر و اگر داشته باشد، ردیاب کار یک وظیفه جدید را به آن اختصاص می‌دهد که از طریق مقدار بازگشتی ضربان به

1 WordCount

آن اطلاع داده می‌شود. معماری کنونی ارتباط یک ردیاب کار واحد با صدها ردیاب وظیفه، منبع بسیاری از مشکلات عملکردی امروزی در خوشه‌های هادوپ بزرگ است. این مقاله برخی از طرح‌های جایگزینی را که جامعه‌ی هادوپ در حال کار بر روی آنها است، با جزییات در بخش‌های آتی شرح می‌دهد.

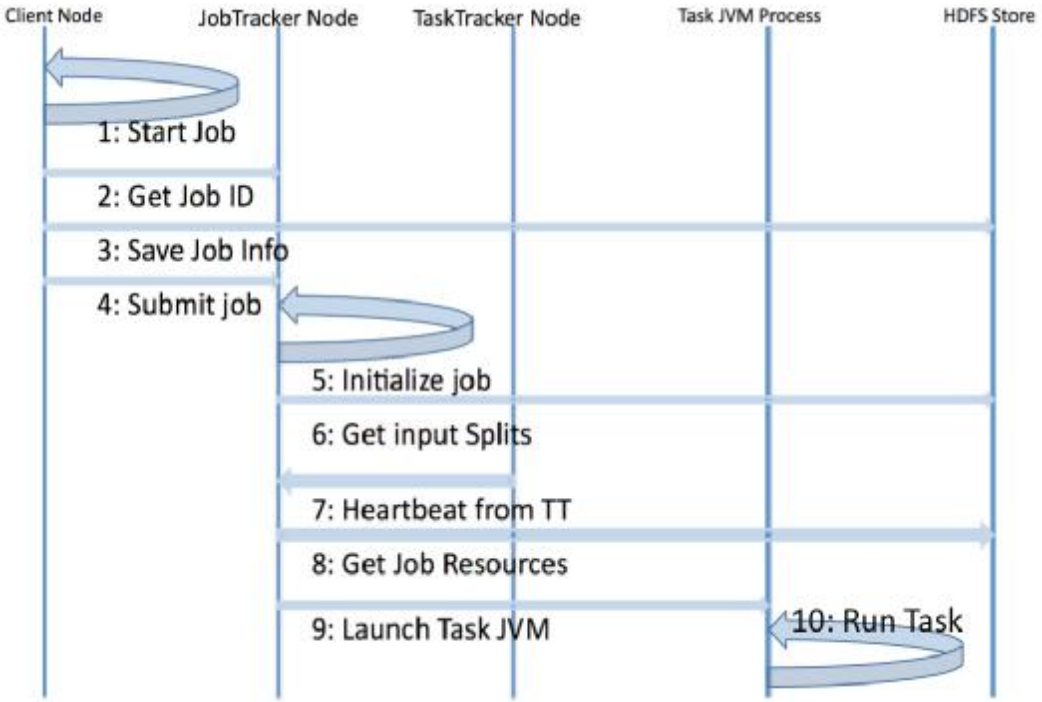
حال که وظیفه‌ای به ردیاب وظیفه محول شده، مرحله بعدی اجرای آن وظیفه به وسیله ردیاب وظیفه است. ابتدا، ردیاب وظیفه فایل JAR کار را، با کپی کردن آن از فایل‌سیستم مشترک به فایل‌سیستم خود، محلی می‌کند. همچنین هر فایل مورد نیازی را از حافظه نهان توزیع‌شده توسط برنامه به دیسک محلی کپی می‌کند. سپس یک دایرکتوری کاری محلی را برای وظیفه ایجاد کرده و محتویات فایل JAR را در این دایرکتوری از حالت jar خارج می‌کند. بعد از آن نمونه‌ای از شیء اجراکننده‌ی وظیفه (TaskRunner) را به منظور اجرای وظیفه ایجاد می‌کند. حال ردیاب وظیفه یک ماشین مجازی جاوا (JVM) را به منظور اجرای هر وظیفه ایجاد می‌کند تا باگ‌های توابع نگاشت و کاهش تعریف‌شده توسط کاربر بر کار ردیاب وظیفه تأثیر نگذارند (یعنی منجر به هنگ یا کرش کردن آن نشوند). البته امکان استفاده مجدد از JVM در بین وظایف مختلف وجود دارد. پردازش فرزند از طریق رابط بطنی با والدش (ضربان قلبی که پیشتر به آن اشاره شد) ارتباط برقرار می‌کند. به این شکل این پردازش هر چند ثانیه والد را در جریان پیشرفت وظیفه قرار می‌دهد تا جایی که وظیفه کامل شود. بسته به این که چارچوب نگاشت‌کاهش با کدام برنامه زمان‌بندی تنظیم شده باشد، می‌توان آن را با یک یا چند صف کار تنظیم کرد. برخی از برنامه‌های زمان‌بندی تنها با یک صف کار می‌کنند، در حالی که برخی دیگر از چندین صف پشتیبانی می‌کنند. می‌توان صف‌ها را با استفاده از پارامترهای مختلف تنظیم کرد[13]. کلاس وضعیت وظیفه (TaskStatus) شمارش وضعیت‌های مختلف یک کار نگاشت‌کاهش را اعلام می‌کند. این فازهای مختلف عبارتند از STARTING؛ MAP؛ SHUFFLE؛ SORT؛ REDUCE و CLEANUP.

الگوریتم 1 در زیر حلقه اصلی ردیاب کار را نشان می‌دهد. ما پیش‌تر در مورد فاز STARTING کار با جزییات زیاد صحبت کرده‌ایم. حال به سراغ دیگر فازها خواهیم رفت.

الگوریتم 1. جریان اصلی ردیاب کار.

```
loop
waitForMessage();
 if msg == HEARTBEAT then
startMapTasks(); {Initialize Map Tasks}
 startReduceTasks(); {Initialize Reduce Tasks}
 elseif msg == MAP TASK DONE  then
 NotifyReduceTasks();{Intermediate Data Fetch}
elseif msg == REDUCE TASK DONE then
if allReduceTasksComplete then
SignalJobComplete();
endif
else
houseKeepingWork();
endif
endloop
```

شکل 3 فراخوان‌های بین اجزای مختلف چارچوب را از زمان آغاز یک کار نگاشت‌کاهش به درخواست کلاینت تا زمان ایجاد یک JVM جدید به منظور اجرای وظیفه، نشان می‌دهد. فعالیت‌های پاک‌سازی در این نمودار دنباله‌ای نشان داده نشده‌اند.



شکل 3. توالی فراخوان برای یک کار هادوپ.

3-2- فاز نگاشت

حال نوبت به بررسی دقیق فاز نگاشت می‌رسد. یک کار نگاشت‌کاهش به منظور خواندن تقسیم‌های داده‌های ورودی از طریق چند وظیفه نگاشت وجود دارد. شیء کار، اطلاعات مربوط به تمام تقسیم‌ها را ایجاد کرده و فایلی را نوشته است که دارای اطلاعات تقسیم‌ها و دیگر اطلاعات مرتبط با کار است تا ردیاب کار بتواند به آن فایل دسترسی داشته باشد. در اصل ردیاب کار مسئول آغاز وظایف برای کار ارائه شده است. در ادامه به بحث در مورد ورودی و خروجی کار می‌پردازیم.

فرمت ورودی (InputFormat) مشخصات ورودی برای یک کار نگاشت‌کاهش را توصیف می‌کند. چارچوب نگاشت‌کاهش بر فرمت ورودی تکیه می‌کند تا:

- مشخصات ورودی برای کار را تأیید کند.
- فایل‌های ورودی را به نمونه‌های تقسیم ورودی (InputSplit) منطقی تقسیم کند، که سپس هرکدام به یک نگاشت‌کننده مجزا سپرده می‌شوند.
- پیاده‌سازی رکوردخوان (RecordReader) مورد استفاده به منظور ایجاد رکوردهای ورودی از تقسیم ورودی منطقی برای پردازش به وسیله نگاشت‌کننده را فراهم کند.

بیشتر برنامه‌های نگاشت‌کاهش کلاس فرمت ورودی خود را پیاده‌سازی می‌کنند اما کلاس پیش‌فرض، «فرمت ورودی متنی» (TextInputFormat) است که فایل‌ها را به چندین خط تقسیم می‌کند. کلیدها موقعیت

فایل را نشان می‌دهند و مقادیر، خط متن را (که توسط «خوراک خط»[1] یا «بازگشت حامل»[2] محدود می‌شوند). تقسیم ورودی، داده‌هایی را که به وسیله یک نگاشت‌کننده واحد پردازش می‌شوند ارائه می‌کند. «تقسیم فایل» (FileSplit)، تقسیم ورودی پیش‌فرض است. تقسیم فایل، $map.input.file را به مسیر فایل ورودی برای تقسیم منطقی تغییر می‌دهد. رکوردخوان جفت‌های <key, value> از یک تقسیم ورودی را می‌خواند. معمولاً، رکوردخوان دیدگاه بایت‌گرای ورودی را، که به وسیله تقسیم ورودی فراهم شده است، تبدیل کرده و یک دیدگاه مبتنی بر رکورد را برای پردازش به پیاده‌سازی‌های mapper ارائه می‌کند. سپس رکوردخوان مسئولیت پردازش مرزهای رکورد را بر عهده گرفته و کلیدها و مقادیر متناسب با دامنه را به وظایف ارایه می‌کند. «فرمت خروجی» (OutputFormat) مشخصات خروجی را برای یک کار نگاشت‌کاهش توصیف می‌کند. چارچوب نگاشت‌کاهش بر فرمت خروجی تکیه می‌کند تا:

- مشخصات خروجی کار را تأیید کند؛ به عنوان مثال، مطمئن شود که دایرکتوری خروجی از قبل وجود نداشته باشد.
- پیاده‌سازی «رکوردنویس» (RecordWriter) مورد استفاده برای نوشتن فایل‌های خروجی کار را فراهم کند. فایل‌های خروجی در فایل‌سیستم هادوپ ذخیره می‌شوند. «فرمت خروجی متنی» (TextOutputFormat)، فرمت خروجی پیش‌فرض است.

اگر دایرکتوری HDFS اشاره‌شده به وسیله پارامتر $mapreduce.job.dir در حالی که کار در فرآیند اجرا بر روی یک خوشه هادوپ است، فهرست شده باشد، ما می‌توانیم فایل‌های زیر را ببینیم.

```
[ mc2233 ] # hadoop f s −l s / tmp / hadoop / mapred / s t a g i n g
/
condor / . s t a g i n g / job 201303040610 0423
Found 4 i t e m s
job 201303040610 0423 / j o b . j a r
job 201303040610 0423 / j o b . s p l i t
job 201303040610 0423 / j o b . s p l i t m e t a i n f o
job 201303040610 0423 / j o b . xml
```

کلاس «تقسیم کار» (JobSplit) کلاس‌های پایه‌ای مرتبط با تقسیم‌های خواندن/نوشتن را به شکل گروه در می‌آورند. اطلاعات تقسیم بر اساس مصرف‌کننده اطلاعات به دو بخش تقسیم می‌شود. این دو بخش فرااطلاعات[3] تقسیم و اطلاعات تقسیم خام هستند. بخش اول به منظور ایجاد ساختارهای داده‌ای محل وظایف به وسیله‌ی ردیاب کار مصرف می‌شود. بخش دوم در زمان اجرا به وسیله‌ی نگاشت‌ها مصرف می‌شود تا نحوه راه‌اندازی فرمت ورودی/رکوردخوان را بدانند تا داده‌ها بتوانند خوانده شده و به متد mapper تغذیه شوند. این تکه‌های اطلاعات در دو فایل جداگانه نوشته می‌شوند که در فهرست دایرکتوری بالا نشان داده شده است. هر پیاده‌سازی تقسیم، کلاس انتزاعی پایه، یعنی تقسیم ورودی، را تعمیم داده و طول و مکان‌های تقسیم را مشخص می‌کند. طول تقسیم برابر است با اندازه داده‌های تقسیم (به بایت)، در حالی که مکان‌ها فهرستی از

1 line feed
2 carriage return
3 meta information

نام گره‌هایی است که در آن‌ها داده‌های تقسیم محلی هستند. مکان‌های تقسیم، روشی برای برنامه زمان‌بندی هستند تا در مورد انتخاب ماشین مورد نیاز برای اجرای این تقسیم تصمیم‌گیری کند. اجرای یک وظیفه نگاشت بر روی گره‌ای که دارای داده‌های خود است یکی از جنبه‌های مهم است چون به این معنا است که نیازی به انتقال داده‌ها بر روی شبکه وجود ندارد. به اینگونه وظایف نگاشت، وظیفه نگاشت داده محلی می‌گویند. محلی بودن می‌تواند بسته به مکانیزم‌های ذخیره‌سازی و راهبرد اجرایی کلی معانی مختلفی داشته باشد. در مورد HDFS، یک تقسیم معمولاً متناظر با یک اندازه بلوک و مکان‌های داده‌های فیزیکی بر روی مجموعه‌ای از ماشین‌ها است که این بلوک به صورت فیزیکی بر روی آنها قرار دارد (و اندازه این مجموعه به وسیله یک ضریب تکرار مشخص می‌شود).

حال بر می‌گردیم به زمانی که ردیاب وظیفه یک فرآیند فرزند را (کلاسی با نام «فرزند»[1]) که دارای یک متد main است آغاز می‌کند. این کلاس مسئول آغاز وظیفه نگاشت است. نام کلاسی که وظیفه نگاشت را در بر دارد «وظیفه نگاشت» (MapTask) است که به تمام جنبه‌های یک نگاشت در حال اجرا برای کار مورد بحث رسیدگی می‌کند (و وظیفه و همچنین کلاس «وظیفه‌ی کاهش»[2] را تعمیم می‌دهد). نگاشت در زمان اجرا در حال جمع‌آوری رکوردهای خروجی در یک بافر درون-حافظه به نام «بافر خروجی نگاشت» (MapOutputBuffer) است. اگر کاهش‌دهنده‌ای وجود نداشته باشد، از یک «جمع‌کننده خروجی نگاشت مستقیم» (DirectMapOutputCollector) استفاده می‌شود که نوشتن را مستقیماً بر روی دیسک انجام می‌دهد. اندازه کل این بافر درون-حافظه‌ای به وسیله ویژگی $io.sort.mb مشخص شده و به طور پیش‌فرض برابر با 100 مگابایت است. از این 100 مگابایت، $io.sort.record.percent به منظور ردیابی مرزهای رکورد حفظ می‌شود. این ویژگی به طور پیش‌فرض برابر با 05/0 است (یعنی 5 مگابایت در حالت پیش‌فرض). هر رکورد مورد ردیابی 16 بایت فضای حافظه را اشغال می‌کند (هرکدام 4 عدد صحیح 4-بایتی) که یعنی با تنظیمات پیش‌فرض بافر می‌تواند 327680 رکورد خروجی نگاشت را ردیابی کند. باقی‌مانده‌حافظه (104/857/600 بایت – (16 بایت × 327/680) = 99/614/720 بایت) به منظور ذخیره‌سازی بایت‌های واقعی جمع‌آوری‌شده استفاده می‌شود (که در حالت پیش‌فرض حدوداً 95 مگابایت خواهد بود). در حین جمع‌آوری خروجی‌های نگاشت، آنها در باقی‌مانده حافظه ذخیره شده و مکانشان هم در بافر درون-حافظه‌ای ردیابی می‌شود. زمانی که یکی از این دو بافر به آستانه مشخص‌شده به وسیله $io.sort.spill.percent که در حالت پیش‌فرض 8/0 (یعنی 80 درصد) است برسد، بافر بر روی دیسک تخلیه می‌شود. برای داده‌های واقعی این مقدار برابر است با 79/691/776 (0/8 × 99/614/720) و برای داده‌های ردیابی برابر است با 262/144 (0/8 × 327/680).

سرریز تمام این اطلاعات بر روی دیسک در یک رشته مجزا انجام می‌شود تا نگاشت بتواند به اجرا شدن خود ادامه دهد. دلیل این که این سرریز زود شروع می‌شود (تنها در زمانی که 80 درصد بافر پر شده است) هم همین است تا پیش از اتمام سرریز به طور کامل پر نشود. اگر یک خروجی نگاشت واحد آنقدر بزرگ باشد که در بافر حافظه جای نگیرد، یک سرریز تنها برای این مقدار صورت می‌گیرد. یک سرریز در حقیقت از یک فایل به ازای هر پارتیشن تشکیل شده است، یعنی یک فایل به ازای هر کاهش‌دهنده.

1 child
2 ReduceTask

پس از پایان یک وظیفه نگاشت، ممکن است چند سرریز بر روی ردیاب وظیفه وجود داشته باشد. این فایل‌ها باید به یک فایل واحد به ازای هر پارتیشن تبدیل شده و سپس به وسیله کاهش‌دهنده‌ها دریافت شوند. ویژگی io.sort.factor$ مشخص می‌کند که هر بار چه تعدادی از این فایل‌ها تبدیل به یک فایل می‌شوند. هرچقدر این عدد کوچکتر باشد، باید از گره‌های بیشتری عبور کرد تا به مقصد رسید. پیش‌فرض این مقدار، 100 است. اگر نگاشت‌کننده‌ها خروجی داده‌ای بسیار زیادی داشته باشند، این ویژگی می‌تواند بسیار تاثیرگذار باشد. این ویژگی به حافظه زیادی نیاز ندارد اما هرچقدر بزرگتر باشد تعداد فایل‌های باز افزایش پیدا می‌کند. به منظور تعیین این مقدار، چند کار نگاشت‌کاهش مورد انتظار در تولید باید اجرا شده و یکی از آنها باید به دقت فایل‌های log را تحت نظر بگیرد. پس تا به این‌جا فایل ورودی تقسیم و خوانده و تابع mapper برای هر جفت <key, value> فراخوانی شده است. پس از پردازش هر جفت، mapper یک جفت < key, value> دیگر را به عنوان خروجی تولید کرده که به وسیله چارچوب در یک بافر حافظه دریافت، مرتب و پارتیشن‌بندی شده است. پس از پر شدن حافظه، این داده‌ها بر روی دیسک سرازیر شده و تبدیل به فایل می‌شدند. تعداد فایل‌ها برابر با تعداد پارتیشن‌های مختلف (یعنی تعداد کاهش‌دهنده‌ها) است. حال وظایف کاهش‌دهنده وارد عمل شده و با یک‌دیگر نحوه دریافت داده‌های خروجی نگاشت به وسیله وظایف کاهش‌دهنده را خواهیم دید[13].

3-3-فاز بُر زدن

فاز «بُر زدن»[1] یکی از مراحلی است که به فاز کاهش یک کار نگاشت‌کاهش می‌انجامد. بازیگران اصلی در فاز بر زدن، وظایف کاهش‌دهنده و ردیاب‌های وظیفه هستند که فایل‌های خروجی نگاشت را در اختیار دارند. به عبارت دیگر، یک فایل خروجی نگاشت بر روی دیسک محلی ردیاب وظیفه‌ای که وظیفه‌ی نگاشت را اجرا کرده است ذخیره می‌شود (دقت کنید که اگرچه خروجی‌های نگاشت همواره بر روی دیسک محلی ردیاب وظیفه نوشته می‌شوند، اما خروجی‌های کاهش ممکن است نوشته نشوند)، اما حالا ردیاب وظیفه‌ای که وظیفه کاهش را بر روی این پارتیشن انجام خواهد داد نیاز به آدرس ردیاب وظیفه‌ای دارد که وظیفه نگاشت را تمام کرده است. علاوه بر این، وظیفه کاهش به خروجی نگاشت پارتیشن مختص به خود از چندین وظیفه نگاشت در سرتاسر خوشه احتیاج دارد. وظایف نگاشت ممکن است در زمان‌های مختلف پایان بپذیرند، بنابراین وظایف کاهش، کپی کردن خروجی‌های‌شان را (که به آن بر زدن خروجی می‌گویند) به محض تمام شدن هرکدام از آنها آغاز می‌کنند. این مرحله همچنین با نام فاز کپی وظیفه کاهش شناخته می‌شود. وظیفه کاهش دارای تعداد رشته‌های کپی‌کننده کمتری است (به طور پیش‌فرض پنج رشته) تا بتواند به طور موازی خروجی‌های نگاشت را هم دریافت کند. با این که مقدار پیش‌فرض 5 است، اما می‌توان این عدد را با تنظیم ویژگی mapred.reduce.parallel.copies$ تغییر داد. از آنجا که، همین‌طور که وظایف نگاشت با موفقیت کامل می‌شوند، کاهش‌دهنده‌ها باید بدانند که خروجی را از کدام ردیاب‌های وظیفه بگیرند، آنها ردیاب وظیفه والد را در مورد به‌روزرسانی وضعیت مطلع می‌کنند، که خود ردیاب کار را مطلع می‌کند. این اعلان‌ها از از طریق مکانیزم ارتباطی ضربان قلب که پیشتر توضیح داده شد مخابره می‌شوند. بنابراین، برای هر کاری، ردیاب کار نگاشت میان خروجی‌های نگاشت و ردیاب‌های وظیفه را می‌داند. یک رشته در کاهش‌دهنده به طور دوره‌ای مکان‌های خروجی‌های نگاشت را از ردیاب کار درخواست می‌کند تا زمانی که همه آنها را دریافت

[1] shuffle

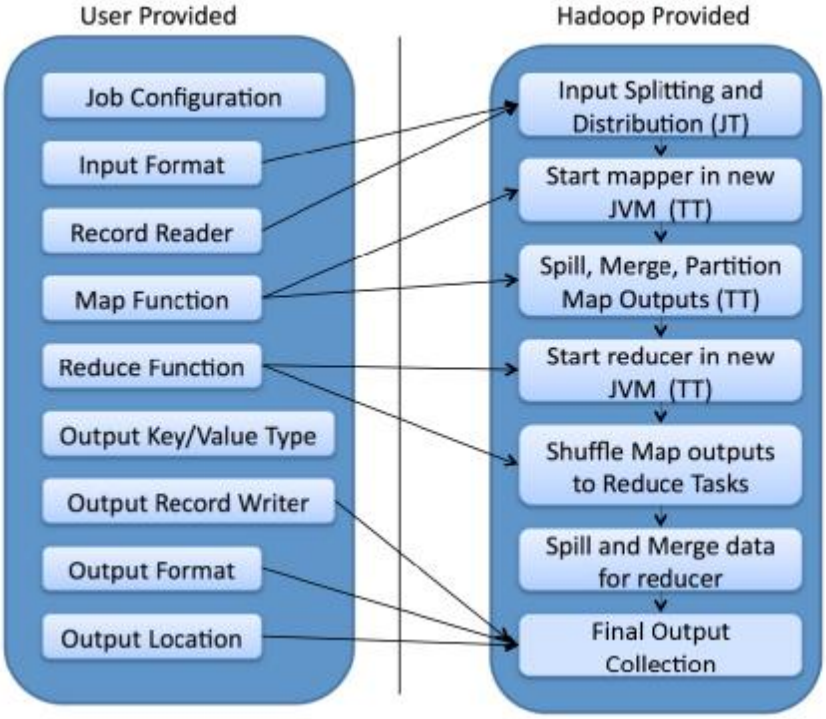
کند. ردیاب‌های وظیفه به محض دریافت شدن خروجی‌های نگاشت به وسیله اولین کاهش‌دهنده آنها را حذف نمی‌کنند چون ممکن است وظیفه کاهش‌دهنده با موفقیت به سرانجام نرسد. در عوض آنها منتظر دستور حذف از سوی ردیاب کار پس از پایان کار می‌مانند. سپس خروجی‌های نگاشت در صورتی که به اندازه کافی کوچک باشند به حافظه کاهش‌دهنده کپی می‌شوند (اندازه‌ی بافر به وسیله $mapred.job.shuffle.input.buffer.percent کنترل می‌شود، که نسبت بخشی را که برای این منظور استفاده می‌شود مشخص می‌کند)؛ در غیر این صورت، آنها بر روی دیسک کپی می‌شوند. زمانی که یک بافر درون-حافظه‌ای به اندازه آستانه (کنترل‌شده به وسیله‌ی $mapred.job.shuffle.merge.percent) یا تعداد آستانه‌ی خروجی‌های نگاشت ($mapred.inmem.merge.threshold) می‌رسد، ادغام شده و بر روی دیسک سرازیر می‌شود[13].

4-3-فاز کاهش

حال دوباره به سراغ بخش کاهش فرآیند می‌رویم. همانطور که کپی‌ها بر روی دیسک جمع می‌شوند، یک رشته پس‌زمینه آنها را ادغام و به فایل‌های بزرگتر تبدیل می‌کند. این عمل منجر به صرفه‌جویی در زمان به منظور ادغام کردن آنها پس از کپی کردن می‌شود. توجه داشته باشید که هر خروجی نگاشت فشرده‌شده (به وسیله وظیفه نگاشت) باید به منظور ادغام شدن در حافظه از حالت فشرده خارج شوند. زمانی که تمام خروجی‌های نگاشت کپی شده‌اند، وظیفه کاهش وارد فاز مرتب‌سازی می‌شود (که در حقیقت باید فاز ادغام نام بگیرد چون مرتب‌سازی در طرف نگاشت انجام شده است)، که خروجی‌های نگاشت را ادغام کرده و ترتیب‌شان را حفظ می‌کند. این عمل در چند دور انجام می‌شود. به عنوان مثال، اگر 100 خروجی نگاشت وجود داشته باشد و ضریب ادغام برابر با ده باشد (مقدار پیش‌فرضی که می‌تواند به وسیله ویژگی $io.sort.factor تنظیم شود، درست مانند ادغام نگاشت)، آنگاه ده دور وجود خواهد داشت. در هر دور ده فایل تبدیل به یک فایل می‌شوند، پس در پایان ده فایل واسطه وجود خواهد داشت. به جای یک دور پایانی که این ده فایل واسطه را تبدیل به یک فایل کند، ادغام به اندازه یک سفر به دیسک و یک I/O اضافی صرفه‌جویی کرده و مستقیماً به تغذیه تابع کاهش در آخرین فاز می‌پردازد: یعنی فاز کاهش. این ادغام پایانی می‌تواند با ترکیبی از قسمت‌های درون حافظه و بر روی دیسک انجام بگیرد. تعداد فایل‌های ادغام‌شده در هر دور در واقعیت کم‌تر از میزانی است که در مثال بالا آورده شده است. هدف، ادغام کردن کمترین تعداد فایل به منظور دستیابی به ضریب ادغام برای دور پایانی است. در جریان فاز کاهش، تابع کاهش برای هر کلید در خروجی مرتب‌شده فراخوانی می‌شود. خروجی این فاز مستقیماً بر روی فایل‌سیستم خروجی که معمولاً HDFS است نوشته می‌شود. در مورد HDFS، از آنجا که گره ردیاب وظیفه یک «گره داده» (DataNode) را هم اجرا می‌کند، نسخه دوم بلوک اول بر روی دیسک محلی نوشته می‌شود. در میان تمام مراحل آغاز و اجرای یک برنامه نگاشت‌کاهش که در بالا شرح داده شدند، برخی از مسئولیت‌ها بر عهده برنامه‌نویس کار خواهد بود. هادوپ باقی امکانات را فراهم می‌آورد. بیشتر کار سنگین به وسیله چارچوب انجام می‌شود. شکل 4 مراحل مختلف انجام‌شده به وسیله کاربران و نحوه نگاشت مراحل مختلف فراهم‌شده به وسیله خود چارچوب توسط کاربران را نشان می‌دهد[13].

3-5- فاز خاموشی

کارهای نگاشت‌کاهش، کارهایی دسته‌ای هستند که در زمانی طولانی اجرا شده و می‌توانند از چند دقیقه تا چند ساعت یا حتی چند روز به طول بیانجامند. از آنجا که این زمان به شکل چشم‌گیری طولانی است، لازم است که کاربر بازخوردهایی در مورد پیشرفت کار دریافت کند. هر کار و وظایف مربوط به آن دارای یک وضعیت هستند، که شامل چیزهایی مانند حالت کار یا وظیفه (مثلاً در حال اجرا، تمام‌شده، لغوشده)، پیشرفت نگاشت‌ها و کاهش‌ها، مقادیر شمارنده‌های کار و یک پیام وضعیت توصیف (که می‌تواند به وسیله کدهای کاربر تنظیم شود) می‌شود. این وضعیت‌ها در طول کار تغییر کرده و این تغییرات به اطلاع کلاینت می‌رسند. یک وظیفه، زمانی که در حال اجرا است، پیشرفت خود، یعنی درصد کامل شدن وظیفه، را ردگیری می‌کند. برای وظایف نگاشت، این درصد برابر است با نسبت ورودی پردازش‌شده است. برای وظایف کاهش سیستم می‌تواند نسبت ورودی کاهش پردازش‌شده را برآورد کند. سیستم این کار را با تقسیم کل پیشرفت به سه بخش متناظر با سه فاز کاهش‌دهنده انجام می‌دهد.



شکل 4. مراحل نگاشت‌کاهش – چه کسی چه کاری را انجام می‌دهد.

سنجش پیشرفت تا حدی گنگ است اما به هادوپ اطلاع می‌دهد که وظیفه در حال انجام کاری است. به عنوان مثال، وظیفه‌ای که در حال نوشتن رکوردهای خروجی است در حال پیشرفت است، اما نمی‌توان آن را در قالب درصدی از تعداد کل رکوردهایی که نوشته خواهند شد نشان داد، چون عدد دوم در اینجا مشخص نیست و اما وظیفه‌ای که خروجی را تولید می‌کند هم آن را نمی‌داند. گزارش دادن پیشرفت مهم است تا هادوپ جلوی وظیفه‌ای که در حال پیشرفت است را نگیرد. تمام عملیات‌های زیر پیشرفت را نشان می‌دهند:

- خواندن یک رکورد ورودی (در یک نگاشت‌کننده یا کاهش‌دهنده)
- نوشتن یک رکورد خروجی (در یک نگاشت‌کننده یا کاهش‌دهنده)

- تدوین توضیحات وضعیت بر روی یک گزارش‌دهنده (با استفاده از متد setStatus گزارش‌دهنده)
- افزایش دادن یک شمارنده (با استفاده از متد incrCounter گزارش‌دهنده)
- فراخوانی متد progress گزارش‌دهنده

وظایف همچنین دارای مجموعه‌ای از شمارنده‌هایی هستند که یا درون چارچوب تعبیه شده‌اند، مانند تعداد رکوردهای خروجی نگاشت نوشته‌شده، یا به وسیله کاربر تعریف شده‌اند، و رویدادهای مختلف را به عنوان اجرای وظیفه می‌شمارند. اگر یک وظیفه پیشرفت خود را گزارش کند، یک پرچم را تنظیم می‌کند تا نشان دهند که تغییر وضعیت باید به ردیاب وظیفه ارسال شود. این پرچم هر چند ثانیه یک بار در یک رشته دیگر بررسی می‌شود و اگر تنظیم شده باشد، وضعیت وظیفه کنونی را به ردیاب وظیفه اطلاع می‌دهد. در همین حال، ردیاب وظیفه در حال ارسال ضربان‌های قبل به ردیاب کار در هر پنج ثانیه است (این میزان، یک میزان حداقل بوده، چون فاصله بین دو ضربان در حقیقت وابسته به اندازه خوشه است: برای خوشه‌های بزرگتر این فاصله بیشتر بوده و این فاصله بیشتر خود منبعی برای نگرانی است چون این مقدار می‌تواند زمان آغاز وظیفه یک کار را به تأخیر بیاندازد)، و وضعیت تمام وظایف در حال اجرا به وسیله ردیاب وظیفه بر روی این فراخوان ارسال می‌شود. شمارنده‌ها معمولاً در فواصل زمانی بیش از پنج ثانیه ارسال می‌شوند چون نسبتاً پهنای باند بیشتری می‌طلبند. ردیاب کار تمام این به‌روزرسانی‌ها را ترکیب کرده تا به یک دید کلی از تمام کارهای در حال اجرا و وظایف تشکیل‌دهنده آنها برسد. در نهایت، کلاینت کار (JobClient) آخرین به‌روزرسانی وضعیت را هر ثانیه یک بار با پرسیدن از ردیاب وظیفه به دست می‌آورد. کلاینت‌ها همچنین می‌توانند از متد getJob کلاینت کار استفاده کرده تا نمونه‌ای از یک شیء کار در حال اجرا (RunningJob) را به دست آورند، که شامل تمام اطلاعات مربوط به وضعیت یک کار است.

زمانی که ردیاب کار، اعلانی را مبتنی بر تمام شدن آخرین وظیفه یک کار دریافت می‌کند، وضعیت کار را به «موفقیت‌آمیز» تغییر می‌دهد. سپس زمانی که کلاینت کار وضعیت را می‌پرسد و در می‌یابد که کار با موفقیت به اتمام رسیده است، پیامی را برای مطلع کردن کاربر چاپ کرده و از متد runJob باز می‌گردد. همچنین در صورتی که تنظیم شده باشد، ردیاب کار یک اعلان کار HTTP را هم می‌فرستد. این بخش می‌تواند به وسیله کاربرانی که می‌خواهند بازخوانی‌ها[1] را هم دریافت کنند با استفاده از ویژگی job.end.notification.url تنظیم شود. در آخر، ردیاب کار حالت در حال کار را برای آن کار پاک‌سازی کرده و به ردیاب‌های وظیفه هم دستور می‌دهد که همین کار را انجام دهند (خروجی واسطه حذف شده و وظایف پاک‌سازی دیگر هم انجام می‌شوند) [13].

4-نوشتن یک برنامه از نگاشت کاهش هادوپ

بهترین راه برای درک و آشنایی با کار هادوپ بررسی یک برنامه نگاشت کاهش هادوپ است. در این بخش، یک برنامه نگاشت کاهش ساده کار به زبان جاوا را بررسی خواهیم کرد که می‌تواند بسیاری از رشته‌ها را معکوس کند. این برنامه از طریق تعدادی از مراحل انجام می‌شود که اول تقسیم داده‌ها به گره‌های مختلف، انجام عملیات به معکوس داده، وابستگی به نتیجه رشته‌ها و سپس بازده نتایج است. این برنامه فرصتی برای بررسی مفاهیم اصلی هادوپ فراهم می‌کند.

1 callbacks

ابتدا نگاهی به بسته‌های و کتابخانه‌های مورد استفاده خواهیم داشت.

reverstringclass در بسته com.javaworld.mapreduce است. می‌توان آن را در مجموعه‌ای از دو ورودی به شرح زیر نشان داد[14]. :

اولین مجموعه‌ی ورودی:

```
package com.javaworld.mapreduce
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.StringTokenizer;
import java.io;*.
import java.net;*.
import java.util.regex.MatchResult
```

دومین مجموعه‌ی ورودی:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapredu.JobClient;
import org.apache.hadoop.mapredu.JobConf;
import org.apache.hadoop.mapredu.MapredBase;
import org.apache.hadoop.mapredu.Mapper;
import org.apache.hadoop.mapredu.OutputCollector;
import org.apache.hadoop.mapredu.Reducer;
import org.apache.hadoop.mapredu.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

اولین مجموعه‌ی ورودی برای طبقه‌های استاندارد جاوا و مجموعه دوم برای اجرای نگاشت کاهش است. reverstringclass با گسترش[1] org.apache.hadoop.conf و پیاده سازی[2] واسط org.apache.hadoop.until.Tool آغاز می‌شود.

نگاشت و کاهش:

دو طبقه درونی برنامه عبارتند از:

- **نگاشت**: شامل قابلیت برای پردازش جفت ورودی کلید- مقدار برای تولید خروجی جفت کلید- مقدار[14].
- **کاهش**: شامل قابلیت جمع‌آوری خروجی از پردازش نگاشت موازی و خروجی‌ای است که داده‌ها جمع آوری می‌کند.

[1] extending
[2] implementing

اکنون، ترکیب تمام خروجی‌های این چینینی لازم است. این کار با استفاده از روش کاهش طبقه کاهش انجام می‌شود همانطور که در مراحل زیر نشان داده شده:

Reduc.redu()

```
public static class Reduc extends MapRedBase
    implements Reducer<Text, Text, Text, Text> {
    public void reduc(Text key, Iterator<Text> values,
                OutputCollector<Text, Text> output,
                Reporter reporter) throws IOException {
      while (values.hasNext()) {
        output.collect(key, values.next());
      }   }       }
```

Map class

```
public static class Map extends MapredBase
      implements  Mapper<LongWritable,  Text,  Text,
Text>
{
  private Text inpText   = new Text();
  private Text reverText = new Text();
  public void map(LongWritable key, Text inputs,
  OutputCollector<Text, Text> output,
  Reporter reporter) throws IOException {
  String inputString = inputs.toString();
  int length = inputString.length();
  StringBuffer reverse = new StringBuffer();
  for(int i=length-1; i>=0; i--)
      {
        reverse.append(inputString.charAt(i));     }
 inputText.set(inpString);
reverseText.set(reverse.toString());
output.collect(inpText,reverText);
 }    }
```

5-چالش‌های زمان‌بندی نگاشت‌کاهش

در هر لحظه صدها کار (کوچک/متوسط/بزرگ) ممکن است بر روی یک خوشه هادوپ برای پردازش قرار داشته باشند. نحوه زمان‌بندی وظایف نگاشت و کاهش این کارها بر روی زمان تکمیل و در نهایت الزامات QoS این کارها اثر می‌گذارد. هادوپ به طور پیش‌فرض از یک برنامه زمان‌بندی FIFO استفاده می‌کند. اما دو برنامه زمان‌بندی دیگر هم ساخته شده‌اند. اولین برنامه Fair Scheduler است که فیسبوک آن را به منظور ارایه زمان پاسخ سریع برای کارهای کوچک و زمان پایان معقول برای کارهای تولیدی طراحی کرده است. دومین برنامه Capacity Scheduler است که توسط یاهو توسعه یافته و صف‌ها را معرفی کرده است که کارها در آنها ارائه می‌شوند. بخشی از کل منبع محاسباتی به صف‌ها تخصیص داده می‌شود و کارها هم دارای اولویت هستند. واضح است که یک نوع الگوریتم و سیاست زمان‌بندی برای تمام انواع کارها مناسب نیست. بسته به ویژگی‌های حجم کاری، ترکیبی از الگوریتم‌های مختلف می‌تواند مناسب‌تر باشد.

اساساً هادوپ یک محصول نرم‌افزاری چندکاره است که می‌تواند چندین مجموعه داده، برای چندین کار و برای چندین کاربر را به طور همزمان پردازش کند. بنابراین زمان‌بندی کارها به نحوی که از منابع موجود بر روی خوشه محاسباتی، بهترین استفاده صورت بگیرد، از دیگر دغدغه‌های هادوپ است. برنامه‌ی زمان‌بندی هادوپ در آغاز کارش، یک برنامه‌ی زمان‌بندی FIFO بود که درون ردیاب کارش تعبیه شده بود. اگرچه بسیار ساده بود اما در پیاده‌سازی انعطاف نداشت و نمی‌شد آن را تنظیم کرد. به هر حال، تمام کارها دارای اولویت یکسان نیستند و یک کار دارای اولویت بالا نباید پشت یک کار دسته‌ای اولویت پایین طولانی منتظر بماند. در سال 2008، هادوپ یک رابط زمان‌بندی قابل اتصال را معرفی کرد که مستقل از ردیاب کار عمل می‌کرد. هدف، ساختن برنامه‌های زمان‌بندی جدیدی بود که به بهینه‌سازی زمان‌بندی بر اساس ویژگی‌های خاص کار کمک کنند. یکی دیگر از مزایای این برنامه زمان‌بندی قابل اتصال این است که امکان آزمون و خطای بیشتر و تهیه برنامه‌های زمان‌بندی تخصصی برای پاسخ به نیازهای انواع مختلف کاربردهای نگاشت‌کاهش هادوپ وجود دارد. ردیاب کار پیش از این که وظیفه‌ای را برای ردیاب وظیفه انتخاب کند، باید کاری را انتخاب کند تا نوع وظیفه مشخص شود. پس از انتخاب یک کار، حال ردیاب کار یک وظیفه را برای کار انتخاب می‌کند. ردیاب‌های وظیفه دارای تعداد ثابتی از اسلات‌ها برای وظایف نگاشت و وظایف کاهش هستند: به عنوان مثال، یک ردیاب وظیفه ممکن است قادر به اجرای همزمان دو وظیفه‌ی نگاشت و دو

وظیفه‌ی کاهش باشد. برنامه زمان‌بندی پیش‌فرض ابتدا اسلات‌های خالی وظیفه نگاشت را پر می‌کند، بنابراین اگر ردیاب وظیفه دارای اسلات خالی وظیفه نگاشت باشد، ردیاب کار یک وظیفه نگاشت را انتخاب می‌کند؛ در غیر این صورت، یک وظیفه کاهش را انتخاب خواهد کرد. بر اساس تحقیقات انجام شده در [16، 15]، کاملاً واضح است که یک برنامه زمان‌بندی واحد برای دستیابی به بهترین QoS ممکن برای یک خوشه نگاشت‌کاهش تحت یک حجم کاری متنوع، کافی نیست. همچنین، ما مشاهده کرده‌ایم که هیچکدام از برنامه‌های زمان‌بندی فشار وارده بر گره‌ها در اثر اجرای وظایف مختلف را در نظر نگرفته‌اند.

5. چالش‌های کارایی نگاشت‌کاهش

صدها هزار کار با مطالبه‌های متفاوت از CPU، I/O و شبکه بر روی خوشه‌های هادوپ متشکل از صدها گره اجرا می‌شوند. وظایف بر روی ماشین‌ها زمان‌بندی می‌شوند که در بیشتر موارد دارای 16 هسته یا بیشتر هستند. کارهای کوتاه دارای الزامات زمان تکمیل متفاوتی نسبت به کارهای طولانی هستند و به همین شکل کارهای اولویت بالای تولیدی نیاز به کیفیت سرویس متفاوتی در مقایسه با کارهای نوع پرسشی لحظه‌ای دارند. پیش‌بینی زمان تکمیل کارهای نگاشت‌کاهش هادوپ یک موضوع تحقیقاتی مهم است چون برای شرکت‌های بزرگ، پیش‌بینی درست زمان تکمیل و زمان‌بندی مؤثر کارهای هادوپ مستقیماً بر درآمدشان تأثیر می‌گذارد. تحقیقات بسیار در زمینه کارایی نگاشت‌کاهش هادوپ انجام می‌شود. در اینجا در مورد برخی از مهم‌ترینِ این تحقیقات که با این مقاله مرتبط هستند صحبت می‌کنیم. در گزارش فنی سال پژوهش [17]، «هرودوتو» (Herodotou) مجموعه‌ای از مدل‌های کارایی را برای توصیف یک کار نگاشت‌کاهش بر روی هادوپ با دقت شرح داد. این مدل‌ها را می‌توان به منظور برآورد کارایی کارهای نگاشت‌کاهش و همچنین یافتن تنظیمات بهینه برای استفاده هنگام اجرای کارها به کار برد. مجموعه‌ای از 100 معادله، هزینه کلی یک کار نگاشت‌کاهش را بر اساس پارامترهای مختلف، مانند پارامترهای هادوپ، آمار پروفایل کار و مجموعه‌ای از پارامترها که هزینه I/O، CPU و شبکه اجرای یک کار را تعریف می‌کنند، محاسبه می‌کند. در مقاله‌ای در سال 2010، Kavulya et al. [18] لاگ‌های نگاشت‌کاهش یاهو برای مدت ده ماه را بررسی شد و از یک روش یادگیری مبتنی بر نمونه که از محلی بودن موقتی بهره می‌برد، استفاده کردند تا زمان‌های تکمیل کار را بر اساس داده‌های تاریخچه پیش‌بینی کنند. اگرچه تمرکز این مقاله بر روی بررسی لاگ‌های یاهو است، اما این مقاله رویکردهای یادگیری مبتنی بر مورد (نزدیک‌ترین همسایه) را گسترش می‌دهد. این الگوریتم ابتدا از یک رویکرد مبتنی بر فاصله برای یافتن مجموعه‌ای از کارهای مشابه در گذشته نزدیک استفاده کرده و سپس مدل‌های رگرسیونی را تولید می‌کند که زمان تکمیل کارهای آتی را از پارامترهای ورودی مختص به هادوپ فهرست‌شده در مجموعه کارهای مشابه پیش‌بینی می‌کند. Verma همکاران [16] یک مدل کارایی را طراحی کردند که مقدار منابع مورد نیاز برای تکمیل کار در مهلت داده‌شده را برای یک کار (با پروفایل معلوم) و SLO آن (مهلت نرم) برآورد می‌کند. آنها همچنین یک برنامه زمان‌بندی مبتنی بر «هدف سطح سرویس» را پیاده‌سازی کردند که ترتیب کارها و مقدار منابع تخصیصی برای رسیدن به مهلت‌های کار را مشخص می‌کند. نویسنده‌ها در [15]، به توصیف یک مدل تحلیلی می‌پردازند که به منظور پیش‌بینی زمان تکمیل فاز نگاشت یک کار نگاشت‌کاهش ساخته‌اند. در این پژوهش اثر فشار بر روی گره‌های محاسباتی یک خوشه را در نظر گرفته و از یک مدل شبکه صف‌بندی بسته [19] به منظور برآورد زمان تکمیل فاز نگاشت یک کار بر

اساس تعداد وظایف نگاشت، تعداد گره‌های محاسباتی، تعداد اسلات‌ها در هر گره و تعداد کل اسلات‌های نگاشت موجود بر روی خوشه استفاده می‌کنند.

6-چالش‌ها و مشکلات امنیتی هادوپ

هادوپ مشکلات امنیتی خاصی را برای مدیران مراکز داده و متخصصین امنیتی ایجاد می‌کند. این ایرادات امنیتی عبارتند از:

1) داده تجزیه شده: خوشه‌های بزرگ داده شامل اطلاعاتی است که به آسانی جابجا می‌شوند و اجازه انتقال کپی‌های مختلف از چندین نقطه را دارا هستند و بدین ترتیب قابلیت افزونگی و انعطاف در داده‌ها را فراهم می‌کنند. داده برای تجزیه در دسترس است و می‌تواند بین چندین سرور به اشتراک گذاشته شود. بنابراین در اثر تجزیه داده‌ها، پیچیدگی بیشتری به کل مجموعه افزوده شده و از آنجا که مدل امنیتی وجود ندارد، خود یک ایراد امنیتی به شمار می‌رود.

2) محاسبات پراکنده: از آنجا که در اختیار داشتن منابع، منجر به پردازش مجازی داده‌های در دسترس می‌شود، این مسئله باعث ایجاد سطح بزرگی از محاسبات موازی می‌شود. بنابراین فضاهای پیچیده‌ای ساخته می‌شوند که در مقایسه با مواقعی که پردازش به صورت متمرکز و یکپارچه صورت می‌گیرد و تامین امنیت آنها آسان‌تر است، دارای ریسک بالاتری در برابر حملات هستند.

3) کنترل دسترسی به اطلاعات: فضاهای داده مورد نظر که دسترسی را در سطح مدل مهیا می‌سازند، در آدرس‌دهی به کاربران پیشنهادی به لحاظ وظایف و دسترسی به سناریوهای مرتبط، عاری از جزئیات دقیق‌تر هستند. در حال حاضر بسیاری از طرح‌های امنیتی پایگاه‌داده دسترسی براساس وظیفه را مهیا می‌سازند.

4) ارتباط گره به گره: یکی از نگرانی‌ها در روش هادوپ و تعداد دیگری از روش‌های مشابه که در این زمینه وجود دارند، این است که ارتباطی ایمن را پیاده‌سازی نمی‌کنند. آنها به جای TCP/IP از RPC استفاده می‌کنند.

5) تعامل سرویس گیرنده: ارتباط سرویس گیرنده با مدیریت منابع و گره‌های داده صورت می‌گیرد. با این وجود مشکلی در اینجا وجود دارد. اگرچه با استفاده از این مدل ارتباطی کارآمد قابل پیاده‌سازی است، اما حفاظت گره‌ها از سرویس گیرنده‌ها و برعکس و همچنین سرورها از گره‌ها به سختی انجام می‌گیرد. سرویس گیرندگانی که اینچنین در معرض خطر هستند، موجب گسترش داده‌ها و یا لینک‌های مخرب در میان سرویس‌ها می‌شوند.

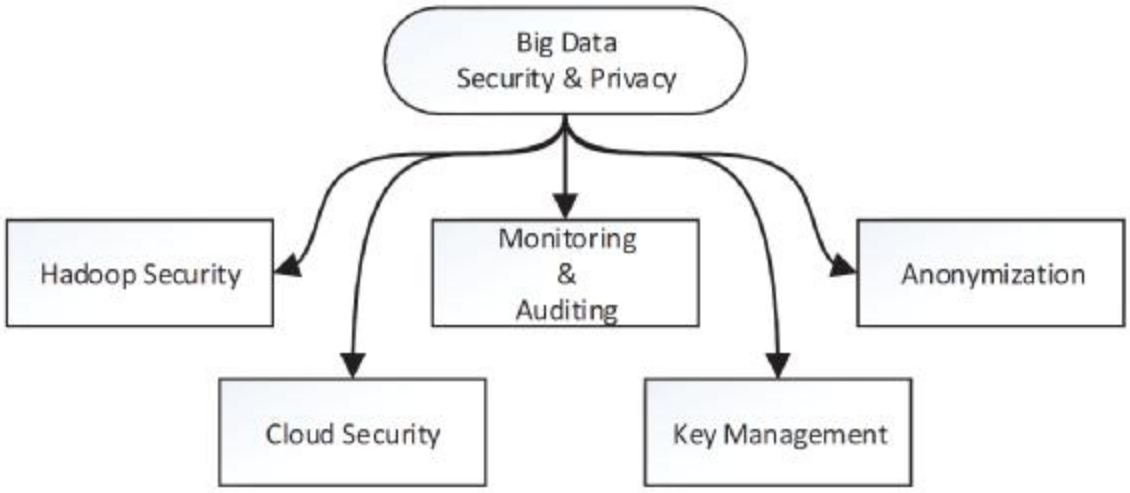
6) تقریباً بدون امنیت: خوشه‌های بزرگ داده بدون در نظر گرفتن امنیت و یا با امنیت ناچیزی طراحی می‌شوند. انتشار و نصب داده‌های کلان بر روی سرویس‌های وب بدون کوچکترین عملی جهت جلوگیری از خطرات احتمالی، آن ها را از این نظر بسیار حساس می‌کند. [20]

برخی از چالش ها در طراحی مکانیزم امنیتی برای هادوپ و بهبود امنیت آن در ادامه معرفی شده اند [21]:

- ابعاد سیستم بزرگ است.
- هادوپ یک سیستم فایل توزیع شده است. بنابراین فایل درخوشه پارتیشن بندی و توزیع شده است.
- فعالیت بعدی ممکن است بر روی یک گره متفاوت از آن چه که کاربر اجازه دسترسی را داده است انجام شود.
- وظایف خواسته شده از سوی کاربران مختلف ممکن است همگی بر روی یک گره انجام شود.

- کاربران می توانند از طریق برخی مسیرها به سیستم دسترسی داشته باشند.

امنیت هادوپ نقش مهمی در امنیت داده‌های کلان برعهده دارد. مطابق با شکل 4، این مسئله آنقدر مهم است که پژوهش [22] امنیت هادوپ را در کنار امنیت ابر، نظارت و ممیزی، مدیریت کلید و ناشناس ماندن، جزء مولفه‌های اصلی امنیت در داده های کلان می‌داند. بر اساس این پژوهش، در هنگام برخورد با داده‌های کلان، راه‌حل‌های امنیتی سنتی برای اطمینان از امنیت و حریم خصوصی داده ها کافی نیست. بلکه باید برای تامین امنیت داده‌های کلان، مجموعه‌ای از راه‌کارهای امنیتی مختلف را استفاده کرد.



شکل 4. دسته بندی حریم خصوصی و امنیت داده های کلان

تاکنون پژوهش‌های مختلفی برای رفع مشکلات امنیتی هادوپ انجام شده است. به عنوان نمونه، پژوهش [23] راهکارهای ذیل را برای امن سازی HDFS پیشنهاد نموده است:

- رویکرد مبتنی برKerberos (پروتکل احراز هویت شبکه)
- رویکرد الگوریتم نقطه‌ی هدف(Bullseye) برای محافظت از اطلاعات حساس
- رویکرد نام گره برای بالا بردن قابلیت دسترسی گره‌ها: این رویکرد از دو گره نام اضافی (Master/Slave) استفاده می‌کند.

8. نتیجه‌گیری

در این مقاله، اکوسیستم هادوپ و نگاشت‌کاهش بررسی و نحوه رسیدگی نگاشت‌کاهش هادوپ به تکمیل یک کار از طریق فازهای مختلف را با جزئیات کامل شرح داده شد. همانطور که ذکر شد این فازها عبارتند از راه‌اندازی، نگاشت، بُر زدن، کاهش و خاموشی. در ادامه با توجه به اینکه بهترین راه برای درک و آشنایی با کار هادوپ بررسی یک برنامه نگاشت کاهش هادوپ است، یک برنامه نگاشت کاهش ساده که می‌تواند رشته-ها را معکوس کند بررسی شد. این برنامه به زبان جاوا نوشته شده و از کتابخانه جاوا مختص کارهای نگاشت کاهش استفاده می‌کند. همچنین در ادامه مقاله به بررسی چالش‌های زمان‌بندی، کارایی و امنیتی نگاشت‌کاهش هادوپ پرداخته شد. همانطور که ذکر شد، یک برنامه زمان‌بندی واحد برای دستیابی به بهترین QoS ممکن برای یک خوشه نگاشت‌کاهش تحت یک حجم کاری متنوع، کافی نیست. همچنین، ما مشاهده کرده‌ایم که هیچ‌کدام از برنامه‌های زمان‌بندی فشار وارده بر گره‌ها در اثر اجرای وظایف مختلف را در نظر

نگرفته‌اند. همچنین ذکر شد که پیش‌بینی زمان تکمیل کارهای نگاشت‌کاهش هادوپ یک موضوع تحقیقاتی مهم است چون برای شرکت‌های بزرگ، پیش‌بینی درست زمان تکمیل و زمان‌بندی مؤثر کارهای هادوپ مستقیماً بر درآمدشان تأثیر می‌گذارد. تحقیقات بسیاری در زمینه کارایی نگاشت‌کاهش هادوپ در حال انجام است. اما این موضوع کماکان به عنوان یک حوزه تحقیقاتی باز مطرح می‌باشد. در نهایت می‌توان ادعا کرد که هادوپ امنیتی قوی در سطح سیستم فایل دارد، اما برای ایمن‌سازی کامل دسترسی به داده توسط کاربران و برنامه های تجاری هوشمند، از سطح جزئیات مورد نیاز پشتیبانی نمی‌کند. این مسئله تشکیلات در صنایعی که امنیت در آنها حائز اهمیت است (مانند سرویس‌های مالی، نظامی، سیستم سلامت و دولت) را وادار می-سازد که بین رها کردن داده‌ها و یا قفل کردن تمامی کاربران یکی را انتخاب کنند.

منابع:

[1] Hadoop, Documentation and open source release, http://hadoop.apache.org/core/
[2] Pig, http://pig.apache.org/
[3] Hive, http://hive.apache.org/
[4] Accumulo, http://accumulo.apache.org/
[5] HBase, http://hbase.apache.org/
[6] Sqoop, http://sqoop.apache.org/
[7] Oozie, http://oozie.apache.org/
[8] Flume, http://flume.apache.org/
[9] http://odbms.org/download/Pro Hadoop Ch. 6.pdf
[10] http://answers.oreilly.com/topic/459-anatomy-of-amapreduce-job-run-with-hadoop/
[11] http://developer.yahoo.com/blogs/hadoop/anatomyhadoop-o-pipeline-428.html
[12] http://hadoop.apache.org/docs/r0.18.3/ mapred tutorial.html
[13] Bardhan, Shouvik, and Daniel A. Menascé (2013). ***The Anatomy of MapReduce Jobs, Scheduling, and Performance Challenges,*** Int. CMG Conference.
[14]Seema, M. and Jha., C. K. (2015). ***MapReduce: Simplified data analysis of Big data***. Procedia Computer Science 57,pp. 563-571.
[15] Bardhan, S. and Menasc´ e, D.A. (2012). ***Queuing Network Models to Predict the Completion Time of the Map Phase of MapReduce Jobs***, CMG Intl. Conf., Las Vegas, pp.3-7.
[16] Verma, A., Cherkasova, L. and Campbell, RoyH. (2011)., ***ARIA:automatic resource inference and allocation for mapreduce environments,*** Proc. 8th ACM Intl. Conf. Autonomic computing, ACM,
[17] Herodotos Herodotou, ***Hadoop performance models***, Cornell University Library, Report Number CS2011-05, arXiv:1106.0940v1 [cs.DC].
[18] Kavulya , S. et. al. (2010). ***An analysis of traces from a production mapreduce cluster***, 10th IEEE/ACM Intl. Conf. Cluster, Cloud and Grid Computing (CCGrid).
[19] E. Krevat, T. Shiran, E. Anderson, J. Tucek, J.J. Wylie, and Gregory R Ganger, ***Applying Performance Models to Understand Data-Intensive Computing Efficiency*** (May 2010). Carnegie-Mellon University, Parallel Data Laboratory, Technical Report CMU-PDL10-108.
[20] Sharma, Priya P., and Navdeti, C. P.(2014):, ***Securing big data hadoop: a review of security issues, threats and solution.,*** Int. J. Comput. Sci. Inf. Technol 5.2 ,pp.2126-2131.
[21] Jam, M.R, et al., ***A survey on security of Hadoop*** (2014). Computer and Knowledge Engineering (ICCKE), 2014 4th International eConference on. IEEE.
[22], Duygu Sinanc,T., Terzi,R. and Sagiroglu,S. (2015). ***A survey on security and privacy issues in big data***, *Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference for*. IEEE.
[23] Saraladevi, B., et al. (2015). ***Big Data and Hadoop-A study in security perspective***, *Procedia computer science* 50,pp. 596-601.