دانشکده فنی و مهندسی

کارشناسی ارشد مهندسی کامپیوتر – نرم‌افزار

گروه مهندسی کامپیوتر و فناوری اطلاعات

ساخت پلتفرم های بهینه برای پردازش دیتاهای بزرگ

واحد درسی:

سمینار

استاد راهنما:

دکتر علی رضوی

دانشجو:

آرزو درویشی

(ش.د: ۹۹۰۱۲۷۶۲۱)

زمستان ۱۴۰۰

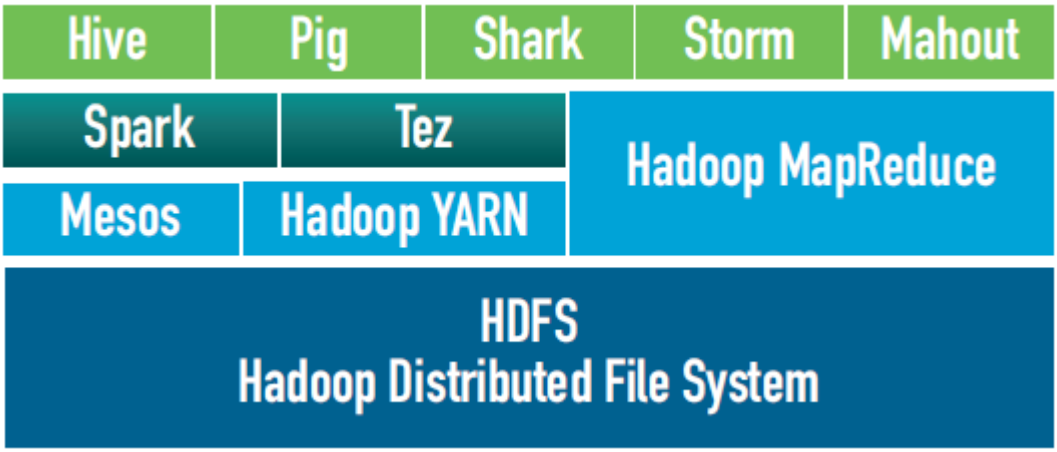
ساخت پلتفرم های بهینه برای پردازش دیتاهای بزرگ

چکیده

ر عصر داده های بزرگ، پلتفرم های خوشه ای و ابتکارات مدیریت منابع برای پاسخگویی به تقاضای رو به رشد برای پردازش مقادیر زیادی داده ایجاد شده اند. یک مجموعه مشترک از وظایف داده های بزرگ شامل چند مرحله است، و هر مرحله به طور معمول یک عملیات داده خاص مانند فیلتر کردن و دسته بندی است. برای موازی کردن اجرای یک کار در یک خوشه، هر مرحله شامل تعدادی از وظایف یکسان است که می تواند به طور همزمان بر روی چندین سرور انجام شود. خوشه های دستی اغلب شامل صدها یا هزاران سرور هستند که تعداد زیادی از وظایف را اداره می کنند. مدیریت منابع که تخصیص منابع و اجرای کار را مدیریت می کند، برای عملکرد سیستم بسیار مهم است. به طور کلی، سه چالش عمده در مدیریت منابع سیستم های جدید پردازش داده های بزرگ وجود دارد. اولاً اگرچه وظایف مختلف از مشاغل و فازهای مختلف معلق هستند، اما با توجه به ویژگی های مختلف وظایف، مانند الزامات منابع و زمان اجرا، تعیین اینکه کدام یک در گرفتن منابع تقدم دارند، دشوار است. دوم اینکه وابستگی بین وظایف وجود دارد که می تواند به طور همزمان انجام شود. برای دو مرحله وظیفه متوالی، خروجی مرحله قبلی ورودی مرحله بعدی است. مدیریت منابع باید با این وابستگی مطابقت داشته باشد. مسئله سوم عملکرد هماهنگ نشده گره های خوشه ای است. در عمل عملکرد زمان اجرا هر سرور متفاوت است. مدیریت منابع باید تخصیص منابع را با توجه به تکامل عملکرد هر سرور به صورت پویا تنظیم کند. مدیریت منابع بر روی پلتفرم های موجود و کارهای قبلی اغلب به پیکربندی های کاربر بستگی دارد و عملکرد سازگاری را بر روی هر گره فرض می کند. با این حال، عملکرد برای انواع حجم کار رضایت بخش نیست. هدف از پایان نامه تحقیق در رویکردهای جدید برای بهبود کارایی پلتفرم های پردازش داده در مقیاس بزرگ است. به طور خاص، هنگامی که سیستم منابع را اختصاص می دهد، عوامل اجرای پویا به دقت در نظر گرفته می شوند. الگوریتم های جدیدی برای جمع آوری داده ها در مورد عملکرد و پیش بینی ویژگی های وظایف و خوشه ها توسعه یافته است. ما همچنین ابتکارات مدیریت منابع را توسعه می دهند که تخصیص منابع را به صورت پویا برای هر مرحله از هر کار انجام شده در خوشه ها تنظیم می کنند. نتایج و روش های جدید این پایان نامه قطعا اطلاعات ارزشمند و الهام بخش در مورد دیگر مسائل مشابه در جامعه تحقیقاتی ارائه خواهد کرد.

مقدمه

در دهه های گذشته شاهد تغییر عمده ای در سیستم عامل های پردازش داده ها بوده ایم، زیرا رشد سریع داده های بزرگ نیازمند کارslaveای بیشتر برای گسترش در خوشه های بزرگ است. تجزیه و تحلیل داده های بزرگ تقریباً در همه جا از زندگی روزمره ما مانند مراقبت های بهداشتی، تجاری، مالی، کنترل ترافیک، تولید و خرده فروشی استفاده می شود. سیستم های کامپیوتری قدرتمند و کارآمد برای پردازش به موقع داده های بزرگ مورد نیاز است. با این حال، سیستم پایگاه داده سنتی مستقر در یک سرور تنها به دلیل افزایش حجم، تنوع، سرعت و صحت، برای رسیدگی به داده های بزرگ ناکافی است. بنابراین، بسیاری از سیستم عاملهای مقیاس پذیر مبتنی بر خوشه توسعه یافته اند تا به تقاضای روزافزون برای پردازش داده های بزرگ به صورت موازی پاسخ دهند. شکل ۱,۱ خانواده رویکردها و مکانیسم های ظهور سیستم های پردازش داده در مقیاس بزرگ را نشان می دهد. در یک خوشه، داده های ورودی و خروجی در یک سیستم فایل توزیع شده مانند HDFS (سیستم فایل توزیع شده Hadoop) ذخیره می شوند. لایه ذخیره سازی توسط سیستم های مختلف محاسبه داده در مقیاس بزرگ مانند MapReduce/Hadoop [2]، Hadoop YARN [3]، Mesos [4]، Tez [5] و Spark [6] مستقر می شوند. مجموعه ای از سیستم های زیست محیطی برای پردازش انواع مختلفی از داده ها و برنامه ها مانند Hive [7]، Pig [8]، Shark [9]، Storm [10] و Mahout [۱۱]] ساخته شده است.



شکل ۱,۱: استقرار معمولی سیستم های محاسبات کلان داده در مقیاس بزرگ

یک مجموعه کلی از کارهای کلان داده شامل دنباله ای از مراحل پردازش است و هر مرحله نمایانگر یک عملیات داده به طور کلی تعریف شده مانند فیلترینگ، ادغام، مرتب سازی و نگاشت است. برای موازی سازی اجرای کار در یک خوشه در مقیاس بزرگ، هر مرحله شامل تعدادی از وظایف یکسان است که می توانند به طور همزمان در چندین سرور اجرا شوند. این تنظیم کلی پردازش داده های چند مرحله ای شامل طیف وسیعی از محصولات تجزیه و تحلیل داده در عمل است. به عنوان مثال، MapReduce/Hadoop یک فرایند معمولی دو مرحله ای را نشان می دهد. سایر برنامه های کاربردی با پردازش داده های چند مرحله ای شامل کارهای زنجیره ای MapReduce برای پرس و جوهای SQL-

on-Hadoop، الگوریتم های یادگیری ماشین تکراری (به عنوان مثال: پیج پیوند، رگرسیون لجستیک، k-mean و سایر موارد در کتابخانه Mahout) و محاسبات علمی (به عنوان مثال: جذب داده ها در پیش بینی آب و هوا و هیدرولوژی GFS و شبیه سازی مبتنی بر معادله دیفرانسیل جزئی) می باشد.

چالش های تحقیق

در این بخش، ما ویژگی های مشترک سیستم های محاسبه داده های بزرگ فعلی و چالش های اصلی تحقیق در مدیریت منابع را بر اساس این ویژگی ها مورد بحث قرار می دهیم. هدف از این کار چالش های زیر است.

۱. حجم های کاری مختلف. سیستم های پردازش داده های بزرگ معمولاً انواع مختلفی از برنامه های ارائه شده توسط کاربران مختلف را همزمان ارائه می دهند. هر کار شامل چندین مرحله است و یک عملیات منحصر به فرد در هر مرحله ارائه می شود. انجام وظایف در مراحل مختلف کارهای مختلف نیازمند منابع متفاوتی است و زمان اجرای متفاوتی دارد. به عنوان مثال، یک کار معمولی MapReduce شامل دو مرحله است: نقشه و کاهش. مرحله نقشه بر پردازش ورودی/خروجی دیسک متمرکز است و مرحله کاهش وظیفه تجزیه و تحلیل داده ها را بر عهده دارد. بنابراین، وظایف در مرحله نقشه معمولاً بیشتر به ورودی/خروجی دیسک متکی هستند و وظایف در مرحله کاهش نیاز به منابع بیشتری در حافظه و CPU دارند. حتی در مرحله کاهش، عملیات مختلف به میزان متفاوتی از منابع متکی است.

۲. وابستگی مراحل. وابستگی معمولاً بین مراحل هر برنامه در سیستم های پردازش داده های بزرگ وجود دارد. برای هر دو مرحله متوالی یک کار، داده های خروجی مرحله قبلی داده های ورودی مرحله بعدی است. از یک سو، مرحله بعدی نمی تواند قبل از مرحله قبلی خود به پایان برسد. مدیریت منابع در خوشه باید با چنین وابستگی مطابقت داشته باشد. از سوی دیگر، مرحله بعدی می تواند زودتر از اتمام مورد قبلی شروع شود. اولین جزء در هر مرحله (به استثنای مرحله اول در یک کار) shuffling نامیده می شود. وظیفه انتقال داده های خروجی تولید شده از مرحله قبلی به خود را بر عهده دارد. شروع زودتر از مرحله shuffling می تواند با تداخل با مرحله shuffling و پایان مرحله قبلی به بهبود عملکرد کمک کند. ما دریافتیم که این دوره منحصر به فرد نقش مهمی در مدیریت منابع ایفا می کند. تعیین مناسب زمان پایان یک مرحله و زمان شروع مرحله بعدی می تواند از منابع بیکار یا داده های انباشته منتظر پردازش جلوگیری کند. با این حال، مراحل مختلف کارهای مختلف اندازه های متفاوتی از داده ها را ایجاد می کند و سرعت انتقال داده ها با توجه به پهنای باند سیستم و تداخل عملیات جابجایی با سایر کارها، همواره در حال تغییر است. بنابراین، برای مدیریت منابع تعیین بهترین زمان برای شروع مراحل بعدی برای کارهای مختلف چالش برانگیز است.

۳. عملکرد ناهماهنگ گره های خوشه ای. در حین اجرای هر خوشه در عمل، عملکرد زمان اجرای هر گره متفاوت است. عوامل زیادی می توانند عملکرد سیستم یک سرور را در خوشه تحت تأثیر قرار دهند، مانند تعداد فرایندهایی که در سرور همزمان اجرا می شود، درصد اشغال حافظه و رقابت عملیات خواندن/نوشتن روی دیسک. این به زمانبند نیاز دارد تا تخصیص منابع را در زمان واقعی با توجه به تغییر عملکرد هر سرور تنظیم کند.

٤. خرابی گره در خوشه. در یک خوشه در مقیاس بزرگ، خرابی گره ها و راه اندازها (سرورهای کند) در عمل طبیعی است. معمولاً تحمل خطا در سیستم های محاسباتی برای رسیدگی به این گونه مسائل اجرا می شود. مکانیسم حدس و گمان یک راه حل رایج برای کاهش تاثیر چنین شکست هایی است. اساساً، هنگامی که یک کلاهبردار شناسایی می شود، یک کپی از وظایف آن (وظایف احتکاری) ایجاد می شود و به سرور دیگری اختصاص داده می شود تا سریعتر به پایان برسد، به طوری که یک کار به کندی وظایف ناشایست کند نمی شود. برای درخواست یک سازوکار سوداگری موثر و کارآمد، سه درخواست باید مورد توجه قرار گیرد. اول، هنگامی که هر گره در خوشه عملکرد متفاوتی دارد، باید بتواند گره گیرها را از گره های آهسته معمولی به طور دقیق تشخیص دهد (به عنوان مثال، یک مکانیسم حدس و گمان باید در یک خوشه ناهمگن به خوبی کار کند). ثانیاً، وقتی یک گره به طور غیرطبیعی کند می شود، مکانیسم حدس و گمان باید این تظاهر کننده را به سرعت تشخیص دهد. سرانجام، پس از شناسایی یک تنگنا، وظایف تکراری آن باید به گره های مناسب با عملکرد خوب اختصاص داده شود، زیرا واگذاری وظایف تکراری به بازکنندگان یا حتی گره های کندتر نمی تواند عملکرد سیستم را بهبود بخشد یا زمان اجرای کار را کاهش دهد. با این حال، ما در می یابیم که سازوکارهای احتکاری موجود نمی توانند این سه خواسته را برآورده کنند.

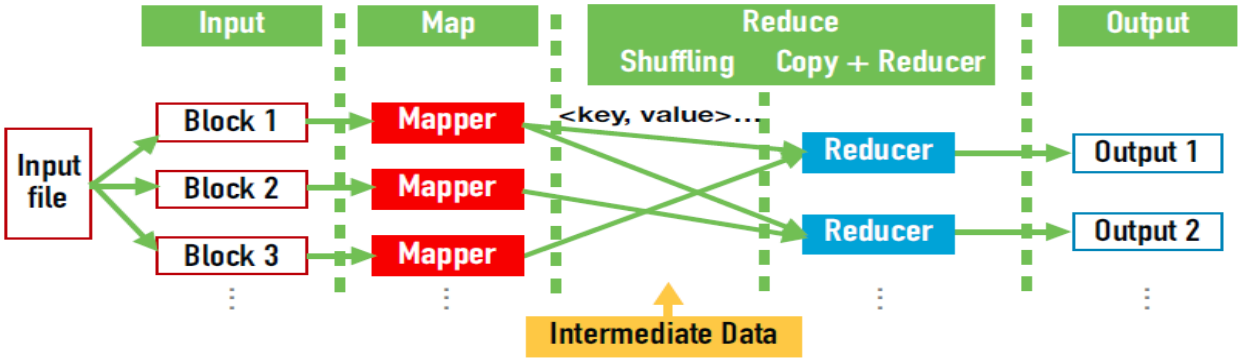
پیش زمینه

ابتدا، ما مدل برنامه نویسی MapReduce، به ویژه جریان کار دو مرحله در یک برنامه MapReduce را معرفی می کنیم. دوم، ساختارهای هر دو پلتفرم های Hadoop و Hadoop YARN به تفصیل توضیح داده شده است. کار ما به عنوان اجزای پلاگین در این سیستم عامل ها اجرا می شود. سرانجام، سیاستهای زمانبندی معمولی درخت، FIFO، Fairو Capacity معرفی شده است. همه آنها زمانبندی پیش فرض در اکثر سیستم های محاسبه داده بزرگ هستند و می توانند در فایل پیکربندی سیستم تنظیم شوند. ما آنها را به عنوان مبنایی برای مقایسه در ارزیابی عملکرد خود در نظر می گیریم.

MapReduce

یک مدل برنامه نویسی است که توسط گوگل برای پردازش مجموعه داده های بزرگ بر روی خوشه های رایانه معرفی شده است .شکل ۲,۱ طرح پردازش داده های موازی MapReduce را نشان می دهد. یک کار معمولی MapReduce شامل دو مرحله است: map و Reduce. هر مرحله شامل چندین کار یکسان است. یکی از وظیفه map (mapper)یک بلوک از داده های خام را که در یک سیستم فایل توزیع شده مشترک ذخیره می شود، پردازش می کند و داده های میانی را به شکل <کلید ، مقدار> تولید می کند. سه مرحله در مرحله Reduce وجود دارد: Copy،Shuffling وReduce

Shuffling . مسئول کپی کردن داده های میانی از مرحله Mapبه مرحله Reduce است. پس از اتمام انتقال همه داده های میانی، مرحله Copy داده ها را جمع آوری می کنند ومرحله Reduce داده های میانی را پردازش می کند و نتایج نهایی را تولید می کند.

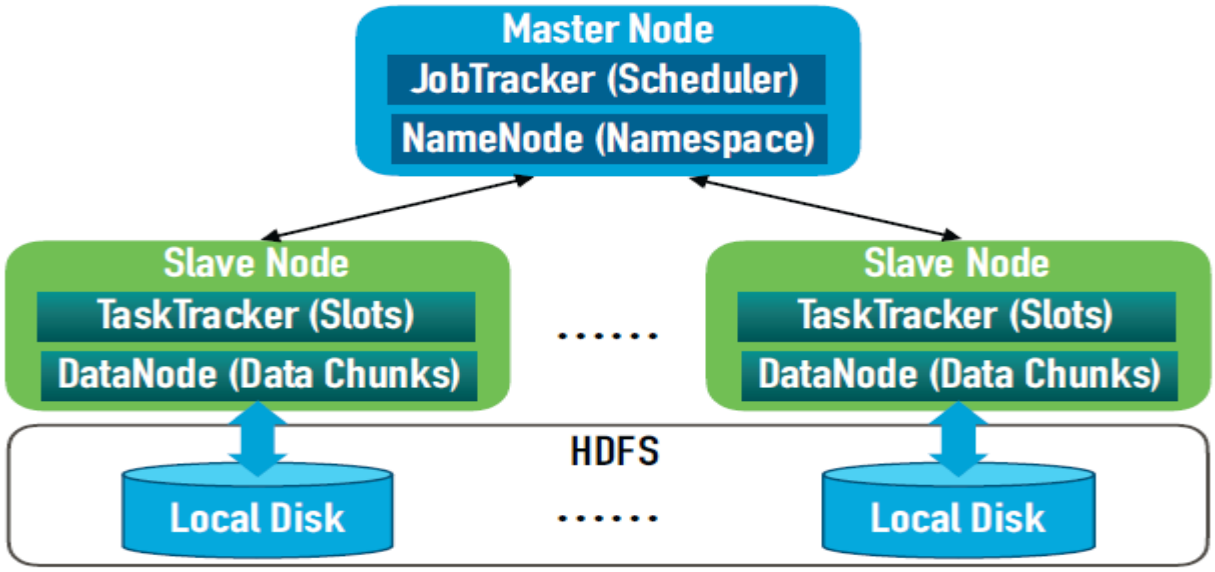


شکل ۲,۱: طرح پردازش داده MapReduce

Hadoop MapReduce

Apache Hadoop MapReduce یک برنامه منبع باز MapReduce است. ساختار Hadoop در شکل ۲,۲ نشان داده شده است. در یک خوشه Hadoop ، یک گره اصلی متمرکز و چندین گره slave توزیع شده وجود دارد. دو جزء اصلی در Hadoop وجود دارد: سیستم فایل توزیع شده Hadoop (HDFS) و سیستم اجرای کار .MapReduce دیسک های محلی از گره های slave به عنوان سیستم فایل توزیع شده با هم ترکیب می شوند. همه داده های ورودی و خروجی به چندین بلوک داده تقسیم شده و جداگانه در آن ذخیره می شوند. هر بلوک داده می تواند چندین کپی اضافی برای تحمل خطا و محل داده داشته باشد NameNode .متمرکز مستقر در گره اصلی مسئول مدیریت HDFSاست. فراداده، نام فایلها و مکانهای بلوک را ذخیره می کند. در هر گره slave، یک ماژول DataNode داده های ذخیره شده را مدیریت می کند.

در اجرای کار، کاربران برنامه ها را به گره اصلی ارسال می کنند. همه مشاغل توسط JobTracker در گره اصلی برنامه ریزی و مدیریت می شوند. یک کار شامل چندین نقشه/کاهش وظایف است. زمانبند در JobTracker وظیفه تعیین وظایف به TaskTracker گره های slave مناسب را دارد و هر TaskTracker مسئول اجرای وظایف است. در هر گره slave، منابع به عنوان اسلات وظیفه نشان داده می شوند. TaskTracker تعدادی از نقشه ها را مدیریت می کند/ اسلات ها را کاهش می دهد که می توانند برای اجرای وظایف نقشه یا کاهش وظایف استفاده شوند. یک اسلات وظیفه می تواند یک کار را در یک زمان اجرا کند.



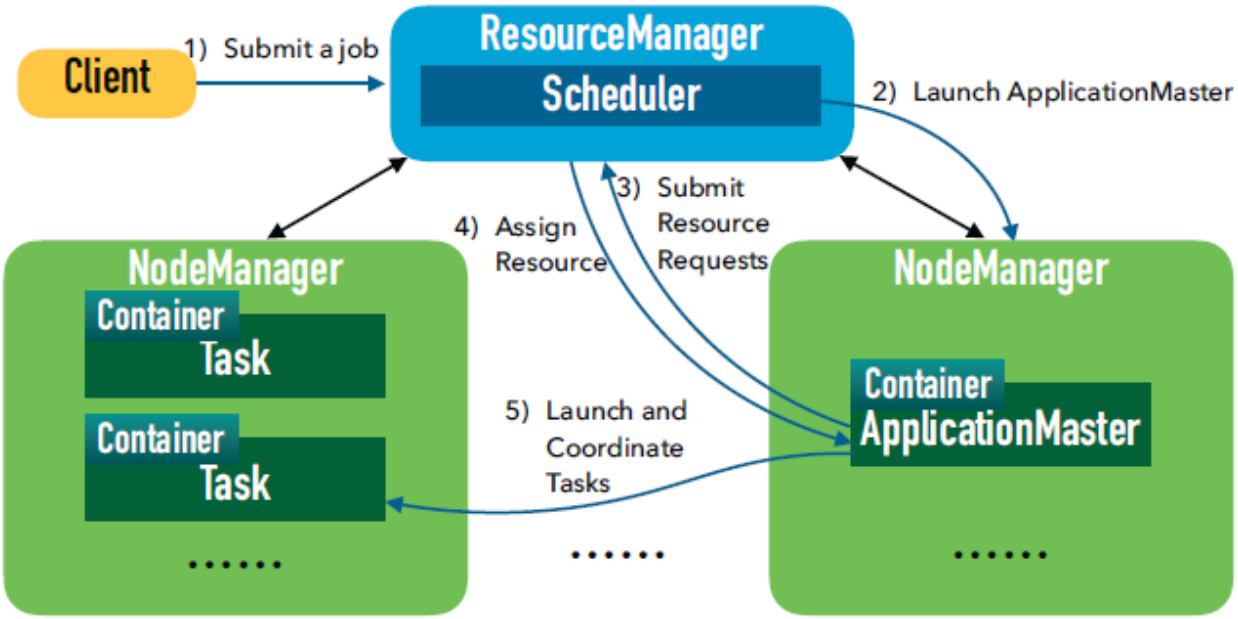
شکل ۲,۲: ساختار Apache Hadoop

وظایف نقشه (کاهش نسبت) فقط می توانند بر روی اسلات های نقشه (کاهش نسبت) انجام شوند. تعداد اسلات های نقشه/کاهش در هر TaskTracker را می توان در فایل پیکربندی سیستم تنظیم کرد. هنگامی که خوشه Hadoop راه اندازی شد، چنین پیکربندی قابل تغییر نیست.

Hadoop YARN

Hadoop YARN نسل بعدی Hadoop است. مشابه Hadoop MapReduce ، در Hadoop YARN، یک گره اصلی متمرکز وجود دارد که ResourceManager را اجرا می کند و چندین گره slave توزیع شده توسط NodeManager روی هر slave مستقر شده است. با این حال، Hadoop YARN دارای دو تفاوت اصلی با نسل اول Hadoop است. اول، YARNاز مدیریت منابع دانه ریز پشتیبانی می کند. هر گره slave ظرفیت منبع را در قالب هسته و حافظه CPU مشخص می کند. هر وظیفه در هنگام ارسال یک تقاضای منابع را مشخص می کند و ResourceManager با اعطای یک منبع در یک گره slave به تقاضای منابع پاسخ می دهد. تفاوت دوم این است که مدیریت منابع و هماهنگی کاری در YARN از هم جدا هستند. یک جزء جدید ApplicationMaster از هماهنگی کار مراقبت می کند. و ResourceManager فقط مسئول زمانبندی است. در این حالت، YARN می تواند از چارچوب های مختلف در یک خوشه مانند MapReduce و Spark پشتیبانی کند.

در شکل ۲,۳ نحوه انجام کار در Hadoop YARN نشان داده شده است. هنگامی که کاربران کاری را به ResourceManager ارسال می کنند، ApplicationMaster در NodeManager راه اندازی می شود. سپس ApplicationMaster منابع مربوط به وظایف کار را به ResourceManager درخواست می کند. ResourceManager به چندین توکن به ظروف پاسخ می دهد. هر توکن منبعی را که یک وظیفه می تواند بدست آورد شرح می دهد. در داخل نشانه ها، Application-Master ظروف را در NodeManager راه اندازی کرده و وظایفی را برای اجرا تعیین می کند.



شکل ۲,۳: اجرای کار در Hadoop YARN

سیاستهای برنامه ریزی رایج

با حجم زیادی از کارهای ارسال شده به صورت دسته ای توسط چندین کاربر، مدیریت منابع و زمان بندی نقش مهمی در عملکرد هر شغل و سیستم کلی ایفا می کند. محققان در برنامه ریزی کار، مدیریت منابع، طراحی برنامه و برنامه های Hadoop تلاشهای فوق العاده ای انجام داده اند [۱۵، ٦ ، ۷ ، ۱۸، ۱۹، ۲۰، ۲۱، ۲۲، ۲۳]. در میان آنها، FIFO، Fair و Capacity نمونه های معمولی هستند که توسط اکثر سیستم های محاسبه داده بزرگ ارائه می شوند.

در زمانبندی FIFO ، همه مشاغل به ترتیب زمان ارسال در صف قرار می گیرند. شغلی که ابتدا ارسال می شود ابتدا ارائه می شود. هنگامی که وظایف آن برای اجرا اختصاص داده شد، کار بعدی در صف اجرا می شود. هدف برنامه ریزی Fairاختصاص منابع عادلانه در مشاغل در طول زمان است. اولویت ها و وزن ها ممکن است برای کارهای مختلف در تخصیص منابع پیکربندی شوند. برنامه ریزی Capacity، عملکردی مشابه برنامه ریزی Fair را ارائه می دهد. در قسمت Capacity، صف های شغلی متعددی برای کاربران مختلف تعریف شده است. زمانبندی منابع یکسانی را برای هر صف ارائه می دهد در حالی که زمانبندی FIFO برای کارها در همان صف ارائه شده است.

سیاست های زمان بندی بالا تنظیمات پیش فرض در اکثر سیستم های محاسباتی مانند Hadoop MapReduce ، Hadoop YARN و Spark است. با این حال، آنها کارآیی استفاده از منابع سیستم را در نظر نمی گیرند.

تخصیص منابع مبتنی بر کار

ما یک طرح مدیریت منابع جدید به نام FRESH را پیشنهاد می کنیم. درFRESH، ما یک مؤلفه نظارتی جدید برای شناسایی بارهای کاری زمان اجرا هر مرحله در خوشه ایجاد می کنیم. بر اساس نتایج نظارت، FRESH به صورت پویا تخصیص منابع را برای مراحل مختلف تنظیم می‌کند تا بهره برداری از منابع سیستم را بهبود بخشد و طول انجام کارهای دسته‌ای را کاهش دهد. علاوه بر این، یک مکانیسم عدالت بهبود یافته در FRESH تضمین می کند که منابع برابر به هر کار در حال اجرا در خوشه اختصاص داده می شود. FRESH در پلتفرم Hadoop MapReduce پیاده سازی شده است اما مکانیسم کلی آن به راحتی در سایر سیستم های محاسباتی قابل استفاده است.

پیشینه و انگیزه

یکی از چالش های بزرگ برای کاربران Hadoop این است که چگونه سیستم های خود را به درستی پیکربندی کنند. Hadoop به عنوان یک سیستم پیچیده با مجموعه بزرگی از پارامترهای سیستم ساخته شده است. در حالی که انعطاف پذیری برای سفارشی کردن یک خوشه Hadoop برای برنامه های مختلف را فراهم می کند، اغلب برای کاربر درک آن پارامترهای سیستم و تعیین مقادیر بهینه برای آنها دشوار است. در این کار، ما یک پارامتر بسیار مهم Hadoop، پیکربندی اسلات را هدف قرار می دهیم و مجموعه ای از راه حل ها را برای بهبود عملکرد، با توجه به گستردگی مجموعه ای از کارها و عدل را در میان آنها، توسعه می دهیم.

در یک خوشه کلاسیک Hadoop، هر کار شامل چندین نقشه و وظایف ساده می شود. مفهوم "اسلات" برای نشان دادن ظرفیت تطبیق وظایف در هر گره در خوشه استفاده می شود. هر گره معمولا دارای تعدادی از پیش تعریف شده اسلات است و یک اسلات می تواند به عنوان یک اسلات نقشه یا یک اسلات ساده پیکربندی شود. نوع اسلات نشان می دهد که کدام نوع وظایف (نقشه یا ساده) می تواند انجام شود. در هر زمان معین، فقط یک کار می تواند در هر اسلات اجرا شود. در حالی که پیکربندی اسلات برای عملکرد بسیار مهم است، Hadoop به طور پیش‌فرض از تعداد ثابت اسلات‌های نقشه استفاده می‌کند و اسلات‌ها را در هر گره در طول عمر یک خوشه ساده می‌کند. مقادیر معمولاً با اعداد اکتشافی بدون در نظر گرفتن ویژگی های شغل تنظیم می شوند. چنین تنظیمات ایستا مطمئناً نمی تواند عملکرد بهینه را برای بارهای کاری متفاوت ارائه دهد. بنابراین، هدف اصلی ما رسیدگی به این موضوع و بهبود عملکرد makepan است. علاوه بر طول عمر، عدالت یکی دیگر از معیارهای عملکردی است که ما در نظر می گیریم. عدالت زمانی حیاتی است که چندین کار به طور همزمان در یک خوشه اجرا شوند. با ویژگی های مختلف، هر کار ممکن است مقدار متفاوتی از منابع سیستم را مصرف کند. بدون برنامه ریزی و مدیریت دقیق، برخی از کارها ممکن است راکت بمانند در حالی که برخی دیگر از مزیت ها استفاده کرده و اجرا را بسیار سریعتر به پایان می رساند. کار قبلی این موضوع را بررسی کرده و راه حل هایی را پیشنهاد کرده است. اما متوجه شدیم که کار قبلی به درستی عدالت را برای این فرآیند MapReduce دو فازی تعریف نکرده

است. در این کار، ما یک تعریف جدید عدالت را ارائه می‌کنیم که مصرف کلی منابع را نشان می‌دهد. راه حل ما همچنین با هدف دستیابی به عدالت خوب در بین تمام مشاغل است.

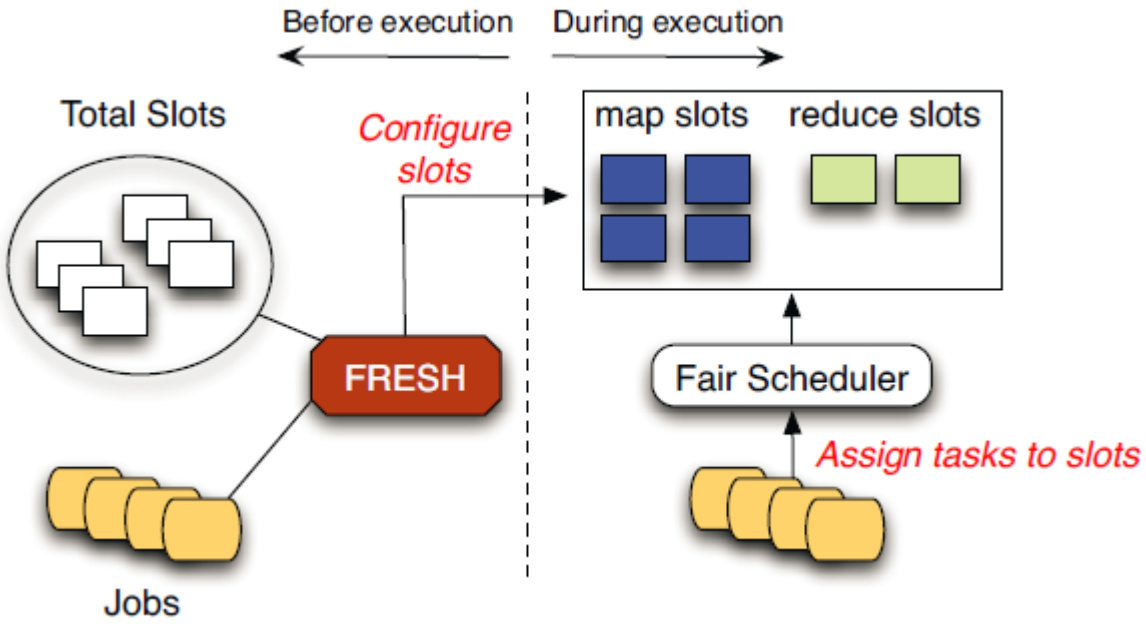
به طور خاص، ما یک رویکرد جدید، "FRESH" را برای دستیابی به پیکربندی و زمان بندی اسلات عادلانه و کارآمد برای خوشه های Hadoop پیشنهاد می کنیم. راه حل ما تلاش می کند تا دو وظیفه اصلی را انجام دهد: (۱) تعیین پیکربندی شکاف، به عنوان مثال، تعداد اسلات نقشه/ساده مناسب است. و (۲) اختصاص نقشه/ساده وظایف به اسلات های موجود. اهداف رویکرد ما عبارتند از به حداقل رساندن زمان ساخت به عنوان هدف اصلی و در عین حال بهبود عدل بدون تنزل دادن makespan. FRESH شامل دو مدل، پیکربندی اسلات استاتیک و پیکربندی اسلات پویا است. در مدل اول، FRESH پیکربندی اسلات را قبل از راه‌اندازی خوشه Hadoop استخراج می‌کند و از همان تنظیمات در طول اجرا درست مانند Hadoop معمولی استفاده می‌کند. در مدل دوم، FRESH به یک شکاف اجازه می دهد تا نوع خود را پس از راه اندازی کلاستر تغییر دهد. هنگامی که یک اسلات وظیفه خود را به پایان می رساند، راه حل ما به صورت پویا اسلات را پیکربندی می کند و وظیفه بعدی را به آن اختصاص می دهد. نتایج تجربی ما نشان می دهد که FRESH به طور قابل توجهی عملکرد را از نظر ساخت و عدالت در سیستم بهبود می بخشد.

فرمول مسأله

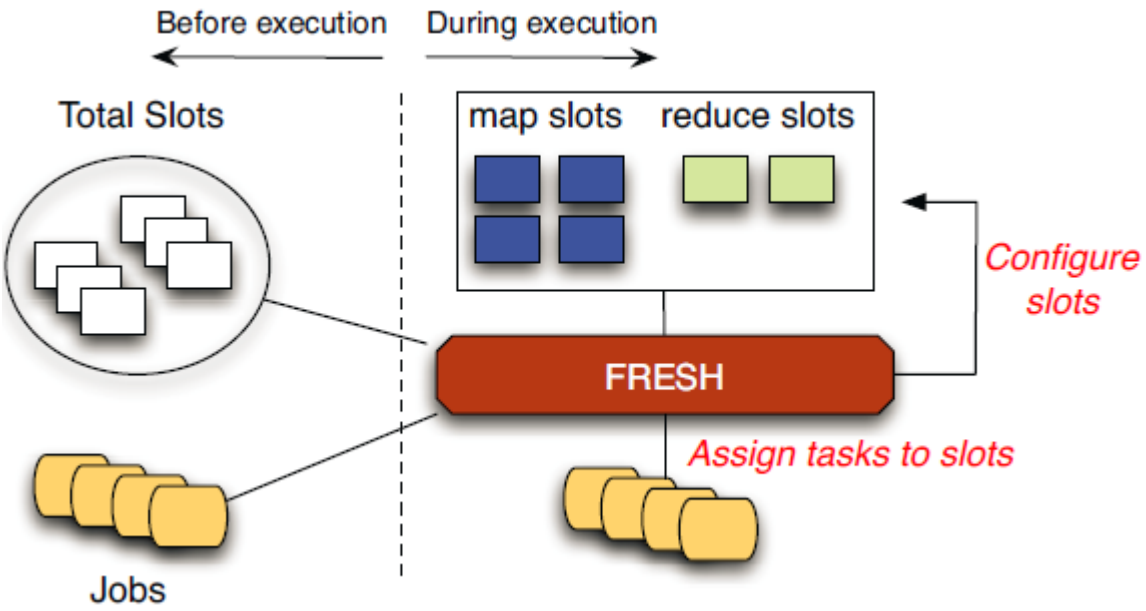
ما در نظر می گیریم که بطور کلی یک کاربر دسته ای از n شغل، J = {J1, J2, . . . Jn}، را به یک خوشه Hadoop با اسلاتهای S ارسال می کند. هر شغل J_i شامل $n_m(i)$ وظایف نقشه و $n_r(i)$ ساده سازی وظایف است. اجازه دهید s_m و s_r تعداد کل اسلات های نقشه باشند و شکاف ها را در خوشه ساده کنید، یعنی $S = s_m+s_r$. ما فرض می کنیم که یک مکانیسم کنترل پذیرش در این خوشه در حال اجرا است به طوری که یک حد بالا در تعداد کارهایی که می توانند همزمان اجرا شوند وجود دارد. به طور خاص، در این شغل فرض می‌کنیم که در هر زمان، حداکثر kشغل در فاز نقشه و حداکثر k شغل در فاز ساده سازی وجود دارد. بنابراین، حداکثر تعداد شغل های فعال در خوشه k، ۲است. در اینجا، k یک پارامتر مشخص شده توسط کاربر برای ایجاد تعادل بین عدل و استاندارد است. هدف ما این است که زمان ساخت (یعنی طول تکمیل کل) مجموعه شغل J را به حداقل برسانیم و در عین حال به تعادل در بین این کارها نیز دست یابیم.

برای حل این مشکل، ما یک راه حل زمان بندی جدید FRESH برای تخصیص اسلات به وظایف Hadoop ایجاد می کنیم. اساساً باید به دو موضوع بپردازیم. اول، با توجه به تعداد کل اسلات ها، نحوه تخصیص آنها برای نقشه و ساده سازی، یعنی تعداد اسلات های نقشه و ساده سازی اسلات مناسب است. دوم، زمانی که یک اسلات در دسترس است، کدام وظیفه باید به آن اختصاص داده شود. FRESH دو مدل را در نظر می گیرد، یعنی پیکربندی اسلات ایستا (نگاه کنید به شکل ۳،۱) و پیکربندی اسلات داینامیک (نگاه کنید به شکل ۳،۲). در مدل اول، تعداد اسلات‌های نقشه و ساده قبل از راه‌اندازی خوشه، مشابه سیستم Hadoop معمولی، تعیین می‌شود. در این مدل، ما فرض می کنیم که پروفایل

های کاری به عنوان دانش قبلی در دسترس هستند. هدف ما این است که بهترین تنظیمات اسلات را بدست آوریم، بنابراین به اولین مسئله رسیدگی کنیم. در طول اجرا، از یک زمان‌بندی عادلانه برای تخصیص وظایف به اسلات‌های موجود استفاده می‌شود. در مدل دوم پیکربندی اسلات پویا، FRESH به یک اسلات اجازه می دهد تا نوع خود را به صورت آنلاین تغییر دهد و بنابراین به صورت پویا تخصیص نقشه و ساده سازی اسلات ها را کنترل می کند. علاوه بر این، FRESH شامل یک الگوریتم است که وظایف را به اسلات های موجود اختصاص می دهد.



شکل ۳,۱: پیکربندی اسلات استاتیک.



شکل ۳,۲: پیکربندی اسلات داینامیک.

راه حل ما: FRESH

در این بخش الگوریتم خود را به صورت FRESH ارائه می کنیم. این شامل دو جزء است: پیکربندی اسلات استاتیک و پیکربندی اسلات داینامیک.

پیکربندی اسلات استاتیک

ابتدا، الگوریتم خود را به صورت FRESH برای پیکربندی اسلات استاتیک ارائه می‌کنیم، جایی که تخصیص نقشه‌ها و اسلات‌های ساده سازی در فایل‌های پیکربندی از پیش تنظیم می‌شوند و هنگام راه‌اندازی خوشه Hadoop بارگذاری می‌شوند. هدف ما این است که پیکربندی بهینه اسلات را با توجه به پروفایل های بار کاری مجموعه ای از کارهای Hadoop بدست آوریم.

ما فرض می کنیم که حجم کار هر شغل به عنوان دانش قبلی در دسترس است. این اطلاعات را می توان از سوابق اجرایی تاریخی یا برآورد تجربی به دست آورد. فرض کنید $\bar{t}_m(i)$ و $\bar{t}_r(i)$ میانگین زمان اجرای یک کار نقشه و یک کار ساده سازی job j_i باشد.

ما $w_m(i)$ و $w_r(i)$ را به‌عنوان بارهای کاری وظایف نقشه و کاهش وظایف j_i تعریف می‌کنیم، که نشان‌دهنده جمع‌بندی زمان اجرای همه وظایف نقشه و ساده سازی وظایف j_i است. بنابراین، $w_m(i)$ و $w_r(i)$ را می توان به صورت زیر تعریف کرد:

$$wm(i) = nm(i)\cdot \bar{t}m(i),\ wr(i) = nr(i)\cdot \bar{t}r(i).$$

اجازه دهید c_m و c_r تعداد اسلات‌هایی را که یک کار می‌تواند برای اجرای نقشه‌اش و ساده سازی وظایف اشغال کند را نشان دهد. به یاد بیاورید که از زمانبندی عادلانه برای تعیین وظایف استفاده می شود و هر کار فعال به طور مساوی برای وظایف خود اسلات اختصاص داده می شود. بعلاوه، تحت خط مشی کنترل پذیرش ما، یک خوشه شلوغ دارای k کار است که به طور همزمان در فاز نقشه و k کار به طور همزمان در فاز ساده سازی اجرا می شود. بنابراین، c_m و c_r را به صورت زیر تعریف می کنیم:

$$= s_m/k,\ c_r = s_r/k.c_m$$

ما یک الگوریتم جدید ایجاد می کنیم (به الگوریتم ۳,۳,۱ مراجعه کنید) تا پیکربندی بهینه اسلات استاتیک را بدست آوریم. ایده اصلی ما این است که تمام تنظیمات ممکن s_m و s_r را برشماریم، و مقدار maksepan را برای هر جفت معین(s_m, s_r) محاسبه کنیم. ما از M و R برای نشان دادن مجموعه کارهایی که در حال حاضر در فاز نقشه و فاز ساده سازی خود در حال اجرا هستند، استفاده می کنیم. هر عنصر در M و Rشاخص کاری کار در حال اجرا مربوطه است. در ابتدا،{} = R و {k ..... ۲، 1=} M، با توجه به تعاریف آنها، هنگامی که job j_i مرحله نقشه خود را به پایان می رساند، شاخص i از M به R منتقل می شود. اندازه های M و R با پارامتر k در حد بالایی قرار دارند. علاوه بر این، ما از W_i

برای نشان دادن حجم کار باقیمانده j_i در فاز فعلی (یا نقشه یا فاز ساده سازی) استفاده می‌کنیم. قبل از اینکه j_i وارد فاز نقشه شود، W_i روی حجم کاری فاز نقشهW_m (i) ، یعنی $W_i = W_m$ (i) تنظیم می شود. در حین اجرا در مرحله نقشه،W_i با توجه به پیشرفت به روز می شود. هنگامی که j_i job مرحله نقشه خود را به پایان می رساند، W_i به حجم کاری خود در مرحله ساده سازی، یعنی $W_i = w_r$ (i) تنظیم می شود.

الگوریتم ۳،۳،۱ جزئیات راه حل ما را ارائه می دهد. حلقه بیرونی تمام پیکربندی‌های اسلات ممکن را برمی‌شمارد (یعنی s_m و s_r). برای هر پیکربندی خاص، ابتدا بارهای کاری نقشه هر کار را محاسبه می کنیم و فازها را کاهش می دهیم، یعنی W_m(i) و w_r (i) ، و مقدار اولیه W_i را تنظیم می کنیم (خطوط ۳-۵ را ببینید). در خط ۶، ما برخی از متغیرهای مهم را مقداردهی اولیه می کنیم، جایی که M و R همانطور که در بالا تعریف شده است، R نشان دهنده لیست مرتبی از کارهای معلق است که فاز نقشه خود را به پایان رسانده اند، اما هنوز وارد فاز ساده سازی خود نشده اند، و T، به صورت صفر مقداردهی شده مدت زمان این مجموعه کارها را ثبت می کند. مؤلفه اصلی الگوریتم حلقه while است (خطوط ۷-۲۲ را ببینید) که فاصله زمانی را محاسبه می کند و زمانی که M و R هر دو خالی باشند به پایان می رسد. در این حلقه، الگوریتم ما ترتیب اجرای همه کارها را تقلید می کند. M و R هر دو دست نخورده باقی می مانند تا زمانی که یکی از کارهای در حال اجرا فاز نقشه فعلی (یا ساده سازی) خود را به پایان برساند. در هر دور اجرا در حلقه while، الگوریتم ما اولین کاری را که وضعیت را تغییر می‌دهد، پیدا می‌کند و سپس مجموعه‌های کار را متناسب با آن به‌روزرسانی می‌کند. این کار هدف بسته به حجم کار باقیمانده و تعداد اسلات های اختصاص داده شده به هر کار می تواند در فاز نقشه یا ساده سازی باشد. در الگوریتم ۳،۳،۱، خطوط ۸ تا ۹ دو شغل (J_u و J_v) را پیدا می‌کنند که کمترین حجم کاری باقیمانده را به ترتیب در M و R دارند. این دو شغل کاندیدایی هستند که ابتدا مراحل فعلی خود را به پایان برسانند. متغیرهایc_m و c_r نشان‌دهنده تعداد اسلات‌هایی است که به هر یک از آنها تحت خط‌مشی زمان‌بندی عادلانه اختصاص داده شده است. بنابراین، زمان اجرای باقیمانده برای J_u و J_v برای تکمیل مراحل خود به ترتیب $\frac{W_u}{c_m}$ و $\frac{W_v}{c_r}$ است

اگر J_u ابتدا فاز نقشه‌اش را تمام کند (مورد خط‌های ۱۰-۱۶)، سپس u را از M حذف می‌کنیم، makespan فعلی را به‌روزرسانی می‌کنیم و حجم کاری باقی‌مانده J_u را بر روی حجم کاری فاز ساده سازی آن تنظیم می‌کنیم (خط ۱۱). ما همچنین بارهای کاری باقیمانده همه کارهای فعال دیگر را در M و R به روز می کنیم (خطوط ۱۲-۱۳). علاوه بر این، الگوریتم یک کار جدید را برای ورود به فاز نقشه خود در خط ۱۴ انتخاب می کند. در نهایت، u را به R اضافه می کنیم تا در صورت عدم رسیدن به محدودیت ظرفیت R، فاز ساده سازی آن شروع شود. در غیر این صورت، u به دنباله لیست معلق R اضافه می شود (خطوط ۱۵-۱۶).

تابع DeductWorkload برای به روز رسانی بارهای کاری باقیمانده برای کارهای فعال در M یا R فراخوانی می شود. همانطور که در زیر نشان داده شده است، ورودی های این تابع شامل یک مجموعه کار A (به عنوان مثال، M، R) و مقدار بار کاری تکمیل شده w است. حجم کار باقیمانده هر کار i در A با کسر w به روز می شود.

```
function DeductWorkload(A, w){
    /*A: a set of job IDs, w: a workload value */
    for i ∈ A do W_i ← W_i − w end }
```

هنگامی که job J_v فاز ساده سازی خود را به پایان رساند (به مورد دیگر در خطوط ۱۷-۲۱ مراجعه کنید)، ما makespan فعلی و همچنین بارهای کاری باقیمانده سایر کارهای فعال در M و R را به روز می کنیم. به طور مشابه، شاخص v از R حذف می شود. اکنون R' خالی نیست، سپس اولین کار در R' به R منتقل می شود. در پایان، در خطوط ۲۲ تا ۲۳، الگوریتم تعیین فاصله کنونی T را با متغیر Opt MS مقایسه می کند که حداقل زمان را دنبال می کند. و در صورت نیاز Opt MS را به روز می کند. یکی دیگر از متغیرهای کمکی Opt SM برای ضبط پیکربندی اسلات مربوطه استفاده می شود. پیچیدگی زمانی الگوریتم ۳،۳،۱ $O(S \cdot N_T^2)$ است، که در آن $=\sum_i (N_m(i) + N_r(i))N_T$ تعداد کل وظایف همه کارها است. در عمل، سربار محاسبه الگوریتم ۳،۳،۱ خیلی کوچک است. به عنوان مثال، با ۵۰۰ اسلات و ۱۰۰ کار که هر کدام دارای ۴۰۰ وظیفه هستند، سربار محاسبات در سرور دسکتاپ با CPU 2.4 گیگاهرتز، ۰،۵۷۸ ثانیه است..

الگوریتم ۳،۳،۱: پیکربندی اسلات استاتیک

1: **for** $s_m = 1$ to S **do**
2: $s_r = S - s_m$
3: **for** $i = 1$ to n **do**
4: $w_m(i) = n_m(i) \cdot \bar{t}_m(i)$, $w_r(i) = n_r(i) \cdot \bar{t}_r(i)$
5: $W_i = w_m(i)$
6: **end for**
7: $M = \{1, 2, \ldots, k\}, R = \{\}, R\_ = \{\}, T = 0$
8: **while** $M \cup R \neq \varphi$ **do**
9: $u = \arg\min_{i \in M} W_i$, $c_m = s_m / |M|$
10: $v = \arg\min_{i \in R} W_i$, $c_r = s_r / |R|$
11: **if** $W_u / c_m < W_v / c_r$
then
12: $M \leftarrow M - u$, $T = T + W_u / c_m$, $W_u = w_r(u)$
13: Deduct Workload(M, $w_m(u)$)
14: Deduct Workload(R, $w_m(u) / c_m \cdot c_r$)
15: pick a new job from J and add its index to M

16: **if** $|R| < k$ **then** $R \leftarrow R + u$
17: **else** add u to the tail of R'
18: **else**
19: $R \leftarrow R \square v$, $T = T + w_{v/cr}$
20: DeductWorkload(R, W_v)
21: DeductWorkload($M, w_{v/cr} \cdot c_m$)
22: **if** $|R'| > 0$ **then** move R' [0] to R
23: **end if**
24: **end while**
25: **if** $T < Opt\_MS$ **then** $Opt\ MS = T, Opt\ SM = s_m$
26: **end for**
27: **return** $Opt\ SM$ and $Opt\ MS$

پیکربندی اسلات داینامیک

سپس به بررسی مدل در FRESH برای پیکربندی اسلات داینامیک می پردازیم. هدف اصلی این مدل، فعال کردن یک شکاف برای تغییر نوع آن (یعنی نقشه برداری یا ساده سازی) پس از راه اندازی خوشه است. برای انجام آن، ما راه حل هایی را برای پیکربندی اسلات ها و اختصاص وظایف به اسلات ها توسعه می دهیم. علاوه بر این، عادلانه مصرف منابع را در میان کارها بازتعریف می کنیم. بنابراین، هدف ما این است که ضمن دستیابی به بهترین عدل، بدون تنزل عملکرد ساخت، طول کار را به حداقل برسانیم. بقیه این بخش به شرح زیر است: ما ابتدا معیار جدید عدل را معرفی می کنیم. سپس الگوریتمی را برای پیکربندی داینامیک نقشه و ساده سازی اسلات ها ارائه می کنیم. در نهایت، نحوه تخصیص وظایف FRESH به شکاف‌های خوشه را توضیح می‌دهیم.

اندازه گیری عدل کلی

عدل یک معیار عملکرد مهم برای طراحی الگوریتم ما است. با این حال، تعریف سنتی عدل به طور دقیق کل منابع مصرفی مشاغل را منعکس نمی کند. در این بخش فرعی، ما یک رویکرد جدید برای کمی کردن اندازه‌گیری عدل ارائه می‌کنیم، جایی که استفاده از منابع را در فرآیند MapReduce تعریف می‌کنیم و از شاخص Jane [۲۴] برای نشان دادن سطح عدل استفاده می‌کنیم.

در یک سیستم Hadoop معمولی، زمان‌بندی منصفانه به طور یکنواخت اسلات‌های نقشه (به‌عنوان ساده) را به کارهای فعال در مراحل نقشه (به‌عنوان ساده) تخصیص می‌دهد. اگرچه عدل در نقشه و فازهای ساده سازی به طور جداگانه حاصل می شود، اما زمانی که منابع (اسلات) مصرف شده در هر دو نقشه و فاز ساده سازی را با هم ترکیب کنیم، عدالت را در بین تمام کارها تضمین نمی کند. برای مثال، فرض کنید یک خوشه دارای ۴ اسلات نقشه و ۴ اسلات ساده سازی است که ۳ کار زیر را اجرا می کنند: J_1 (۲ کار نقشه و ۹ کار ساده)، J_2 (۳ کار نقشه و ۴ کار ساده) و J_3 (۷ کار نقشه و ۳ کار ساده سازی وظایف). فرض کنید هر کار را می توان در یک واحد زمان تمام کرد. جدول زیر تخصیص

اسلات با زمان‌بندی عادلانه را در ابتدای هر نقطه زمانی نشان می‌دهد («M» و «R» نوع اسلات تخصیص داده شده برای کارها را نشان می‌دهد). در نهایت هر سه کار در ۵ واحد زمانی به پایان می رسد. با این حال، آنها به ترتیب ۱۱، ۷ و ۱۰ اسلات را اشغال می کنند.

	0	1	2	3	4
J_1	2(M)	4(R)	2(R)	1(R)	2(R)
J_2	1(M)	2(M)	2(R)	1(R)	1(R)
J_3	1(M)	2(M)	4(M)	2(R)	1(R)

در این کار، یک معیار عدل جدید به نام عدل کلی را به صورت زیر تعریف می کنیم. در هر نقطه زمانی T، اجازه دهید J’ مجموعه کارهای فعال فعلی در سیستم را نشان دهد و T_i نشان دهنده زمان شروع کار J_i در J’ است. ما از دو ماتریس $[i, jt_m]$ و $[i, jt_r]$ برای نشان دادن زمان‌های اجرای تکلیف نقشه j -th و j -th ام ساده سازی کار sJ_i' استفاده می‌کنیم. توجه داشته باشید که این دو ماتریس شامل کارهای ناتمام است. بنابراین، منابع مصرف شده توسط job J_i با زمان T را می توان به صورت بیان کرد:

$$r_i(T) = \frac{\sum_j t_m[i,j] + \sum_j t_r[i,j]}{T - T_i}.$$

که در آن فرمول بالا منابع مؤثری را نشان می دهد که J_i در طول دوره T- T_i مصرف کرده است. هر چه(Tr_i) بزرگتر باشد، منابع بیشتری به J_i اختصاص داده شده است. علاوه بر این، ما از شاخص Jane روی r_i برای نشان دادن انصاف کلی((F(T))در نقطه زمانی T استفاده می کنیم، به عنوان مثال،

$$F(T) = \frac{(\sum_i r_i(T))^2}{|J'| \sum_i r_i^2(T)},$$

جایی که F(T) ∈ [1/|J’|, 1] و مقدار بزرگتر نشان دهنده عدل بهتر است.

پیکربندی اسلات ها

عملکرد پیکربندی اسلات ها این است که تصمیم بگیرید که چه تعداد اسلات باید بر اساس وضعیت فعلی وظایف نقشه را انجام دهند/ساده کنند. به طور خاص، هنگامی که یک کار تمام شد و یک اسلات آزاد شد، سیستم ما باید نوع این اسلات موجود را تعیین کند تا بتواند وظایف دیگر را انجام دهد. در این بخش، الگوریتمی را به صورت FRESH ارائه می کنیم که به طور مناسب نقشه را پیکربندی می کند و اسلات ها را ساده می کند.

اول از همه، راه حل ما از اطلاعات آماری وظایف تمام شده هر کار استفاده می کند. این اطلاعات در متغیرهای سیستم Hadoop و فایل های گزارش موجود است. فرض کنید ($i\ t^-_m$) و($i\ t^-_r$) به ترتیب میانگین زمان اجرای تکلیف کار J_i's map و کاهش وظیفه باشند. هنگامی که یک کار کامل شد، می توانیم به زمان اجرای آن دسترسی داشته باشیم و سپس ($i\ t^-_m$) یا($i\ t^-_r$) را برای کار J_i که آن وظیفه خاص به آن تعلق دارد، به روز کنیم. علاوه بر این، از ($i\ n'_m$) و($i\ n'_r$) برای نشان دادن تعداد کارهای باقی مانده نقشه و ساده سازی وظایف به ترتیب در کار J_i استفاده می کنیم. حجم کار باقیمانده یک کار J_i سپس به صورت زیر تعریف می شود: $w'_m(i) = n'_m(i) \cdot t^-_m(i), w'_r(i)\ w'_m$(i)، که در آن w'_m(i) حجم کار نقشه باقیمانده J_i و $w'_r(i) = n'_r(i) \cdot t^-_r(i)$ حجم کار کاهش باقیمانده J_i است. در نهایت، مجموع بارهای کاری باقیمانده از همه نقشه های معلق را تخمین زده و وظایف را ساده می کنیم. اجازه دهید RW_m مجموع تمام بارهای کاری نقشه باقی مانده از کارها را در M نشان دهد در حالی که RW_r نشان دهنده مجموع تمام بارهای کاری ساده باقی مانده برای مشاغل در R و R' است. RW_m و RW_r را می توان به صورت زیر محاسبه کرد:

$$RW_m = \sum_{i \in \mathcal{M}} w'_m(i), \quad RW_r = \sum_{i \in \mathcal{R} \bigcup \mathcal{R}'} w'_r(i).$$

توجه داشته باشید که RW_m شامل کارهایی است که در فازهای نقشه خود در حال اجرا هستند (در M) در حالی که RW_r شامل کارهایی است که در مراحل ساده سازی آنها (در R) و همچنین کارهایی که مراحل نقشه خود را به پایان رسانده اند اما منتظر اجرای فاز ساده سازی خود هستند (در R').

شهود الگوریتم در FRESH این است که مشاغل را در نقشه خود نگه می دارد و مراحل را در یک پیشرفت ثابت کاهش می دهد تا بتوان همه کارها را به درستی خط لوله کرد تا از انتظار برای اسلات ها یا داشتن اسلات بیکار اجتناب شود. بنابراین، تعداد اسلات‌های نقشه و کاهش باید متناسب با کل بارهای کاری باقیمانده RW_m و RW_r باشد، یعنی فاز بارگذاری شده سنگین‌تر، اسلات‌های بیشتری را برای انجام وظایف خود دریافت می‌کند. با این حال، این ایده ممکن است در فرآیند کاهشی نقشه به خوبی کار نکند زیرا ممکن است یک تغییر ناگهانی در بارهای کاری باقیمانده رخ دهد. مشکل زمانی ایجاد می شود که یک کار فاز نقشه خود را به پایان می رساند و وارد فاز کاهشی می شود. بر اساس تعریف RW_m و RW_r، این کار کاهش حجم کاری خود را به RWr و یک کار جدید که فاز نقشه خود را شروع می کند و سپس حجم کاری نقشه جدید خود را به RW_m اضافه می کند. با این حال، چنین به‌روزرسانی‌هایی می‌توانند وزنRW_m و RW_r را تا حد زیادی تغییر دهند. برای مثال، اگر $RW_r RW_m$ >> باشد، بیشتر اسلات ها اسلات نقشه هستند. فرض کنید یک کار کم‌فشار به‌تازگی فاز نقشه‌اش را به پایان می‌رساند و حجم کاری زیادی را کاهش می‌دهد، سیستم شکاف‌های کاهشی کافی برای انجام وظایف کاهش جدید را ندارد. مدتی طول می‌کشد تا خوشه با این تغییر حجم کاری ناگهانی سازگار شود، زیرا باید منتظر تکمیل بسیاری از وظایف نقشه قبل از پیکربندی آن اسلات‌های آزاد شده به عنوان اسلات‌های کاهشی باشد.

ما الگوریتم ۳،۳،۲ را برای استخراج پیکربندی بهینه اسلات در حالت آنلاین توسعه می‌دهیم. ما از اصل طراحی اولیه با یک کنترل مبتنی بر آستانه پیروی می کنیم تا اثرات منفی ناشی از تغییرات ناگهانی در نقشه را کاهش دهیم / بار کاری را کاهش دهیم. هنگامی که یک کار نقشه/کاهش به پایان رسید، الگوریتم زمان اجرای کار را جمع‌آوری می‌کند و مجموعه‌ای از اطلاعات آماری شامل میانگین زمان اجرای کار، تعداد وظایف باقی‌مانده، حجم کار باقی‌مانده از کار j_i، و کل بارهای کاری باقی‌مانده را به‌روزرسانی می‌کند. خطوط ۱-۴). به دنبال آن، الگوریتم تابعی به نام CalExpSm را فراخوانی می کند تا تعداد مورد انتظار اسلات نقشه ($expS_m$) را بر اساس اطلاعات آماری فعلی محاسبه کند. اگر انتظار بیشتر از تعداد اسلات های فعلی نقشه (S_m) باشد، این اسلات رایگان به اسلات نقشه تبدیل می شود. در غیر این صورت، ما آن را به عنوان یک اسلات کاهش تنظیم می کنیم.

Algorithm 3.3.2: Configure a Free Slot

```
1: if a map task of job J_i is finished then
2:     update t̄_m(i), n'_m(i), w'_m(i) and RW_m
3: end if
4: if a reduce task of job J_i is finished then
5:     update t̄_r(i), n'_r(i), w'_r(i) and RW_r
6: end if
7: expSm =CalExpSm()
8: if expSm > s_m then set the slot to be a map slot
9: else set the slot to be a reduce slot
```

جزئیات تابع CalExpSm در الگوریتم ۳،۳،۳ ارائه شده است. ما از θcur برای نشان دادن نسبت مورد انتظار اسلات نقشه بر اساس حجم کار باقیمانده فعلی استفاده می کنیم. در خط ۲، یک کار فعال Ja را انتخاب می کنیم که دارای حداقل حجم کار نقشه باشد، یعنی کار Ja ابتدا فاز نقشه خود را به پایان می رساند. اگر Ja هنوز تا اتمام فاز نقشه‌اش فاصله داشته باشد، خطر تغییر بار کاری ناگهانی کم است و تابع فقط θcur ·S را به عنوان تعداد مورد انتظار اسلات نقشه برمی‌گرداند. ما یک پارامتر τ1 را به عنوان آستانه پیشرفت تنظیم می کنیم. زمانی که job Ja به پایان فاز نقشه خود نزدیک می شود، یعنی پیشرفت از τ1 فراتر می رود، تابع مشکل بالقوه با تغییر ناگهانی حجم کار را در نظر می گیرد (خطوط ۱۳-۶). اساساً، این تابع سعی می کند نقشه را تخمین بزند و زمانی که Ja وارد فاز کاهش می شود، بار کاری را کاهش دهد و نسبت مورد انتظار اسلات نقشه θexp را در آن نقطه محاسبه کند. یک تغییر ناگهانی بار کاری زمانی اتفاق می‌افتد که θexp با نسبت اسلات‌های نقشه θcur که بر اساس پیکربندی فعلی دریافت می‌کنیم، کاملاً متفاوت باشد. در صورتی که یک تغییر ناگهانی پیش‌بینی شود، به جای آن از θexp ·S به عنوان راهنمای پیکربندی شکاف جدید استفاده می‌کنیم. در غیر این صورت، تابع همچنان θcur · S را برمی گرداند.

به طور خاص در الگوریتم ۳،۳،۳، زمانی که Ja حجم کار نقشه باقیمانده خود را w' m(a) تمام می کند، فرض می کنیم کارهای دیگر در M تقریباً همان پیشرفت را داشته اند. بنابراین حجم کار نقشه کل به میزان w' m(a) · k کاهش می یابد. سپس یک کار جدید به مجموعه M

می‌پیوندد، اجازه دهید job Jb باشد، و wm(b) به کل حجم کار باقی‌مانده RWm اضافه می‌شود (خط ۷). در همین حال، Ja به مجموعه R یا R’ تعلق دارد و حجم کاری کاهش یافته wr(i) به کل RWr کاهش بار باقیمانده اضافه می شود (خط ۸). متغیر θexp در خط ۹ نشان دهنده نسبت مورد انتظار اسلات نقشه در آن نقطه است. در مرحله بعد، زمانی که Ja فاز نقشه خود را به پایان می رساند، تابع تعداد اسلات های نقشه را به دنبال نسبت پیکربندی θcur تخمین می زند. این شامل تعداد اسلات های آزاد شده از نقطه فعلی است. واضح است که هیچ اسلات نقشه دیگری بر اساس تعریف Ja منتشر نشده است، بنابراین ما فقط باید تعداد اسلات های کاهش موجود را در این دوره تخمین بزنیم. در الگوریتم ۳،۳،۳، ما از η برای نشان دادن این عدد و تخمین مقدار آن بر اساس قضیه ۳،۳،۱ زیر استفاده می کنیم. در خط ۱۱، تعداد اسلات های نقشه s’ m را با استفاده از نسبت پیکربندی فعلی پیش بینی می کنیم، به عنوان مثال، اسلات η · θcur به شکاف های نقشه تبدیل می شوند. در نهایت، تابع نسبت تخمینی را با نسبت مورد انتظار در خط ۱۲ مقایسه می کند. اگر اختلاف بیش از آستانه τ2 باشد، نسبت مورد انتظار آینده θexp را در نظر خواهیم گرفت. در غیر این صورت، ما به استفاده از نسبت پیکربندی فعلی θcur ادامه خواهیم داد.

قضیه ۳،۳،۱ فرض کنید وظایف کاهش با نرخ یک در r واحد زمان به پایان می رسد، تعداد (η) اسلات های کاهش موجود زمانی که Ja حجم کار نقشه باقی مانده خود را به پایان می رساند برابر است با:

Algorithm 3.3.3: Function $CalExpSm()$

1: $\theta_{cur} = \frac{RW_m}{RW_m + RW_r}$
2: $a = \arg\min_{i \in \mathcal{M}} w'_m(i)$
3: **if** the progress of job $J_a < \tau_1$ **then**
4: return $\theta_{cur} \cdot S$
5: **else**
6: Let job J_b be the next that will start its map phase
7: $RW'_m = RW_m - w'_m(a) \cdot k + w_m(b)$
8: $RW'_r = RW_r + w_r(a)$
9: $\theta_{exp} = RW'_m / (RW'_r + RW'_m)$
10: calculate η using Theorem 3.3.1
11: $s'_m = s_m + \theta_{cur} \cdot \eta$
12: **if** $\frac{|\frac{s'_m}{S} - \theta_{exp}|}{\theta_{exp}} > \tau_2$ **then return** $\theta_{exp} \cdot S$
13: **else return** $\theta_{cur} \cdot S$
14: **end if**

$$\eta = \frac{\sqrt{m_a^2 + 4 \cdot c \cdot w'_m(a)} - m_a}{2 \cdot c \cdot r}.$$

که در آن c = θcur/ 2·r·k، w’ m(a) حجم کار نقشه باقیمانده Ja را نشان می دهد و ma تعداد اسلات های نقشه اختصاص داده شده به Ja است.

اثبات فرض کنید کار Ja حجم کار نقشه باقیمانده خود را w m(a) در واحدهای زمانی x به پایان می رساند. طبق فرض ما، هر واحد زمان r یک کار کاهش به پایان می رسد. بنابراین، هنگامی که Ja فاز نقشه خود را به پایان می‌رساند، شکاف‌های کاهش‌دهنده η = x r نیز آزاد می‌شوند. فرض کنید که از θcur برای تخصیص اسلات ها استفاده می کنیم و k کارها در M به طور مساوی به اسلات های تازه منتشر شده اختصاص داده شده اند. Job Ja به طور مداوم اسلات های نقشه جدید x·θcur/ r·k را بدست می آورد. معادل داشتن نیمی از آنها x·θcur/ ۲·r·k از ابتدا است. بنابراین، زمان باقی مانده برای Ja تا پایان فاز نقشه را می توان به صورت زیر تخمین زد:

$$x = w'_m(a)/(\frac{x \cdot \theta_{cur}}{2 \cdot r \cdot k} + m_a),$$

که در آن ma تعداد اسلات های نقشه است که در حال حاضر به Ja اختصاص داده شده است. با حل معادله فوق داریم

$$x = \frac{\sqrt{m_a^2 + 4 \cdot c \cdot w'_m(a)} - m_a}{2 \cdot c}.$$

در نتیجه:

$$\eta = \frac{\sqrt{m_a^2 + 4 \cdot c \cdot w'_m(a)} - m_a}{2 \cdot c \cdot r}.$$

وظایف را به اسلات ها اختصاص دهید

هنگامی که نوع اسلات آزاد شده مشخص شد، FRESH وظیفه ای را به آن اسلات اختصاص می دهد. اساساً، ما باید یک کار فعال را انتخاب کنیم و اجازه دهیم اسلات نقشه/کاهش موجود، یک کار نقشه/کاهش از آن کار را انجام دهد. در FRESH، ما از ایده اصلی در زمان‌بندی منصفانه پیروی می‌کنیم، اما به جای آن از معیار انصاف کلی جدید استفاده می‌کنیم: مصرف منابع را برای هر شغل بر اساس معادله (۳,۱) محاسبه کنید و شغلی را با بیشترین کمبود انصاف کلی انتخاب کنید.

Algorithm 3.3.4: Assign a Task to a Slot

1: Initial: $\mathcal{C} = \{\}$, $now \leftarrow$ current time in system
2: **if** the slot is configured for map tasks **then** $\mathcal{C} \leftarrow \mathcal{M}$
3: **else** $\mathcal{C} \leftarrow \mathcal{R}$
4: **for** each job $J_i \in \mathcal{C}$ **do**
5: $total_i = 0$
6: **for** each task j in job J_i **do**
7: **if** task j is finished **then** $e_j \leftarrow f_j - s_j$
8: **else if** task j is running **then** $e_j \leftarrow now - s_j$
9: **else** $e_j \leftarrow 0$
10: $total_i = total_i + e_j$
11: **end for**
12: $r_i = \frac{total_i}{now - T_i}$
13: **end for**
14: $s = \arg\min_{i \in \mathcal{A}} r_i$
15: assign a task of job J_s to the slot

الگوریتم ۳،۳،۴ راه حل ما را در FRESH برای تخصیص یک کار به یک اسلات موجود نشان می دهد. ما از C برای نشان دادن مجموعه ای از مشاغل نامزد استفاده می کنیم. در ابتدا، C خالی است و ما از متغیر now برای نشان دادن زمان فعلی استفاده می کنیم. هنگامی که یک شکاف برای انجام وظایف نقشه پیکربندی شده است (با استفاده از الگوریتم ۳،۳،۲)، C یک کپی از M است. در غیر این صورت، C یک کپی از R است (خطوط ۲-۳). سپس حلقه بیرونی (خطوط ۴-۱۱) مصرف منابع را برای هر کار در C در زمان جاری محاسبه می کند. متغیر totali که به صورت ۰ در خط ۵ مقداردهی شده است، برای ثبت کل زمان اجرا برای تمام کارهای تکمیل شده و در حال اجرا در job Ji (خطوط ۶-۱۰) استفاده می شود. ما از ej برای نشان دادن زمان اجرای وظیفه j استفاده می کنیم. اگر کار j تمام شده باشد، زمان اجرای آن ej تفاوت بین زمان پایان آن fj و زمان شروع آن sj است (خط ۷). اگر وظیفه j همچنان در حال اجرا است، ej برابر است با زمان فعلی که اکنون با زمان شروع آن sj کسر می شود (خط ۸). هنگامی که کل زمان اجرای کار Ji به دست آمد، همانطور که در خط ۱۱ نشان داده شده است، با نرمال کردن کل زمان اجرای کل بر اساس مدت زمان بین زمان جاری و زمان شروع کار Ji (Ti)، منابع مصرفی ri را بدست می آوریم. شغلی با حداقل مصرف منابع انتخاب می شود تا توسط اسلات موجود خدمت رسانی شود.

ارزیابی عملکرد

در این بخش، عملکرد FRESH را ارزیابی کرده و آن را با سایر طرح‌های جایگزین مقایسه می‌کنیم. ما از FRESH-static و FRESH-dynamic به ترتیب برای نمایش پیکربندی اسلات استاتیک و پیکربندی اسلات پویا استفاده می کنیم.

راه اندازی آزمایشی و بارهای کاری

ابتدا جزئیات پیاده سازی، تنظیمات خوشه و حجم کار برای ارزیابی را معرفی می کنیم.

پیاده سازی

ما FRESH را روی Hadoop نسخه ۰,۲۰,۲ پیاده سازی کرده ایم. برای FRESH-static، ما یک برنامه خارجی برای استخراج بهترین تنظیمات اسلات و اعمال آن در فایل پیکربندی Hadoop ایجاد می کنیم. خود سیستم Hadoop برای FRESHstatic اصلاح نشده است. برای پیاده سازی FRESH-dynamic، چند کامپوننت جدید به Hadoop اضافه کرده ایم. ابتدا، ما سیاست کنترل پذیرش را با پارامتر k پیاده‌سازی می‌کنیم، به‌عنوان مثال، حداکثر k کار مجاز است به طور همزمان در فاز نقشه یا در فاز کاهش اجرا شود. دوم، ما دو ماژول جدید در JobTracker ایجاد می کنیم. یک ماژول اطلاعات آماری مانند میانگین زمان اجرای یک کار را به روز می کند و حجم کار باقی مانده از هر کار فعال را تخمین می زند. ماژول دیگر به گونه ای طراحی شده است که یک اسلات آزاد را به عنوان یک اسلات نقشه یا یک شکاف کاهشی پیکربندی کند و طبق الگوریتم های بخش ۳,۳,۲، وظیفه ای را به آن اختصاص دهد. دو پارامتر آستانه در الگوریتم ۳,۳,۳ به صورت $\tau 1 = 0.8$ و $\tau 2 = 0.6$ تنظیم شده است. ما با مقادیر مختلف آزمایش کرده‌ایم و دریافتیم که عملکرد زمانی نزدیک است که $\tau 1 \in [0.7$ و ۰,۹]، $\tau 2 \in [0.5$ و ۰,۷]. به دلیل محدودیت صفحه، بحث در مورد این دو مقدار اکتشافی را حذف می کنیم. علاوه بر این، پروفایل های شغلی (زمان اجرای وظایف) بر اساس نتایج تجربی زمانی که یک کار به صورت جداگانه اجرا می شود، تولید می شود. با این حال، ما به طور تصادفی ٪۳۰± سوگیری را برای زمان اجرای اندازه‌گیری شده معرفی می‌کنیم و از آنها به عنوان پروفایل‌های شغلی که تخمین‌های تقریبی را نشان می‌دهند، استفاده می‌کنیم.

خوشه Hadoop

تمام آزمایش ها بر روی پلت فرم محاسبات ابری Amazon AWS انجام می شود. ما یک خوشه Hadoop متشکل از ۱۱ m1.xlarge نمونه EC2 آمازون [۲۵]، یک گره اصلی و ۱۰ گره برده ایجاد می کنیم. هر گره باید دارای ۴ اسلات باشد زیرا یک نمونه m1.xlarge در Amazon EC2 دارای ۴ هسته مجازی است. در کل، $S = 40$ اسلات در خوشه وجود دارد. علاوه بر این، برای نشان دادن مقیاس پذیری FRESH، شکل ۳,۱۰ مجموعه ای از آزمایش ها را در یک خوشه بزرگتر نشان می دهد که گره های برده را دو برابر می کند، به عنوان مثال، یک گره اصلی و ۲۰ گره برده با $S = 80$ اسلات به طور کلی.

حجم های کاری

بارهای کاری ما برای ارزیابی، معیارهای کلی Hadoop با مجموعه داده های بزرگ را به عنوان ورودی در نظر می گیرند. به طور خاص، از چهار مجموعه داده در آزمایش‌های ما استفاده می‌شود که شامل داده‌های پیوندهای دسته ویکی ۴/۸ گیگابایتی، و داده‌های رتبه‌بندی فیلم ۴ گیگابایتی/۸ گیگابایتی است. داده‌های ویکی شامل اطلاعات مربوط به دسته‌های صفحه ویکی است و داده‌های رتبه‌بندی فیلم اطلاعات رتبه‌بندی کاربر است. ما شش معیار Hadoop زیر را از [26] Purdue MapReduce Benchmarks Suite برای ارزیابی عملکرد انتخاب می‌کنیم.

- طبقه بندی: داده های رتبه بندی فیلم را به عنوان ورودی در نظر بگیرید و فیلم ها را بر اساس رتبه بندی آنها طبقه بندی کنید.

- نمایه معکوس: فهرستی از اسناد ویکی‌پدیا را به‌عنوان ورودی بگیرید و نمایه‌سازی کلمه به سند ایجاد کنید.
- Wordcount: فهرستی از اسناد ویکی پدیا را به عنوان ورودی بگیرید و تعداد تکرار هر کلمه را بشمارید.
- Grep: فهرستی از اسناد ویکی‌پدیا را به عنوان ورودی بگیرید و یک الگو را در فایل‌ها جستجو کنید.
- رتبه بندی هیستوگرام: یک هیستوگرام از داده های رتبه بندی فیلم (با ۵ bins) ایجاد کنید.
- فیلم های هیستوگرام: یک هیستوگرام از داده های رتبه بندی فیلم (با ۸ bins) ایجاد کنید.

ارزیابی عملکرد

در این بخش، عملکرد FRESH را ارائه کرده و با راه حل های دیگر مقایسه می کنیم. با توجه به دسته‌ای از کارهای MapReduce، معیارهای اصلی عملکرد ما زمان ساخت است، یعنی زمان پایان آخرین کار، و انصاف در بین همه کارها. ما عمدتاً با سیستم Hadoop معمولی با زمان‌بندی Fair و پیکربندی اسلات استاتیک مقایسه می‌کنیم. در تنظیمات ما، هر اسلیو دارای ۴ اسلات است، بنابراین در Hadoop معمولی سه تنظیمات ثابت وجود دارد، ۱ اسلات نقشه / ۳ اسلات های کاهش، ۲ اسلات نقشه / ۲ اسلات های کاهش، و ۳ اسلات های نقشه / ۱ اسلات کاهش. ما به ترتیب از Fair-1:3، Fair-2:2 و Fair-3:1 برای نمایش این سه تنظیمات استفاده می کنیم.

ما دو دسته از آزمایش‌ها را با بار کاری متفاوت انجام داده‌ایم، حجم‌های کاری ساده از یک نوع کار (انتخاب شده از شش معیار MapReduce) و بارهای کاری ترکیبی مجموعه‌ای از کارهای ترکیبی را نشان می‌دهند. در بازنشانی این بخش، نتایج ارزیابی را با این دو دسته از بار کاری به طور جداگانه ارائه می کنیم.

عملکرد با بارهای کاری ساده

برای آزمایش بارهای کاری ساده، ۱۰ کار Hadoop برای هر یک از ۶ معیار بالا ایجاد می کنیم. هر مجموعه از ۱۰ کار، مجموعه داده های ورودی یکسانی را به اشتراک می گذارد و به طور متوالی با فاصله زمانی ۲ ثانیه به خوشه Hadoop ارسال می شود. علاوه بر این، ما مقادیر مختلف k را برای نشان دادن تأثیر سیاست کنترل پذیرش، به ویژه $k = 1,3,5,10$ آزمایش کرده‌ایم.

شکل ۳،۳ عملکرد makespan را نشان می دهد. اول، مشاهده می‌کنیم که در هادوپ معمولی، بهترین عملکرد بیشتر زمانی به دست می‌آید که $k=1$ باشد، یعنی فقط یک کار در نقشه و فاز کاهش که معادل زمان‌بندی FIFO (First-In-First-Out) است. این نشان می‌دهد که در حالی که انصاف را بهبود می‌بخشد، زمان‌بندی نمایشگاه، طول عمر مجموعه‌ای از کارها را قربانی می‌کند. علت اصلی اختلاف منابع بین کارها هست که اجرای هر کار را طولانی می کند. راه حل ما، FRESH-static بدتر از بهترین تنظیمات با زمانبندی Fair نیست. در برخی از حجم کار مانند "طبقه بندی"، بهبود کافی است. علاوه بر این، FRESH-dynamic همیشه بهبود قابل توجهی را در تمام تنظیمات آزمایش شده به همراه دارد. به عنوان مثال، FRESH-Dynamic در مقایسه با زمانبندی Fair با نسبت اسلات ۲:۲ به طور متوسط حدود ۳۲،۷۵٪ در زمان ساخت بهبود می یابد. علاوه بر این، پیکربندی اسلات دینامیکی ما اثر مقادیر مختلف k را کاهش می‌دهد، بنابراین منحنی نسبتاً مسطحی را در شکل ۳،۳ نگه می‌دارد.

شکل ۳,۴ انصاف کلی طراحی شده در بخش را نشان می دهد. ۳,۲ در طول آزمایش های مشابه. در بیشتر موارد، ارزش منصفانه یک تابع افزایشی بر روی k است. به خصوص هنگامی که $k = 10$، که در آن همه مشاغل مجاز به اجرای همزمان هستند (بدون کنترل پذیرش)، تقریباً همه تنظیمات ارزش انصاف خوبی را به دست می آورند. از آنجایی که همه کارها در حجم کاری ساده از یک معیار هستند، فازهای نقشه خود را به صورت موجی تمام می کنند و تقریباً در همان زمان وارد فاز کاهش می شوند. بنابراین زمانبندی منصفانه در این مورد به خوبی عمل می کند (k=10). به طور کلی، FRESH-Dynamic از همه طرح‌های دیگر بهتر عمل می‌کند و حتی زمانی که $k = 3$ یا $k = 5$ باشد، به ارزش عادلانه بسیار نزدیک به ۱ دست می‌یابد.

عملکرد با بارهای کاری مختلط

علاوه بر این، ما عملکرد را با بارهای کاری ترکیبی متشکل از معیارهای مختلف ارزیابی می کنیم. ما به طور خاص پنج مجموعه (مجموعه A تا مجموعه E) از کارهای مختلط را تشکیل می دهیم که جزئیات آنها هنگام ارائه نتایج ارزیابی معرفی می شود. برای بارهای کاری مختلط، ترتیب کارها تأثیر زیادی بر عملکرد دارد. در آزمایش‌های خود، پس از تولید همه کارها، الگوریتم جانسون [۲۷] را انجام می‌دهیم که می‌تواند یک برنامه کاری دو مرحله‌ای بهینه بسازد تا ترتیب را تعیین کنیم، یعنی حجم کار برای ارزیابی فهرستی از مشاغل ترکیبی است.

حجم کار مختلط ما به شرح زیر است. مجموعه A~C شامل تعداد مساوی کار از هر معیار است. جزئیات مجموعه A در جدول ۳.۱ زیر فهرست شده است. از هر معیار دو کار دارد، یک کار از مجموعه داده۴ GB و دیگری از مجموعه داده ۸ GB استفاده می کند. مجموعه B حجم کاری کمتری است، با یک کار از هر معیار، و همه کارها از مجموعه داده ۸ GB استفاده می کنند. علاوه بر این، مجموعه C حجم کاری زیادی را نشان می دهد که حجم کاری مجموعه A را دو برابر می کند، یعنی ۴ کار از هر معیار.

جدول ۳,۱: مجموعه A: 12 کار مختلط

JobID	Benchmark	Dataset	Map #	Reduce #
01	Classification	8GB Movie Rating	270	250
02	Classification	4GB Movie Rating	129	120
03	Invertedindex	8GB Wikipedia	256	250
04	Invertedindex	4GB Wikipedia	128	120
05	Wordcount	8GB Wikipedia	256	200
06	Wordcount	4GB Wikipedia	128	100
07	Grep[a-g][a-z]*	8GB Wikipedia	270	250
08	Grep[a-g][a-z]*	4GB Wikipedia	128	100
09	Histogram_ratings	8GB Movie Rating	270	250
10	Histogram_ratings	4GB Movie Rating	129	120
11	Histogram_movies	8GB Movie Rating	270	200
12	Histogram_movies	4GB Movie Rating	129	100

علاوه بر این، ما دو مجموعه دیگر از کارها (مجموعه D و مجموعه E) ایجاد می کنیم تا حجم کاری فشرده و کم حجم کاری را نشان دهد. بر اساس آزمایش‌های ما با بارهای کاری ساده، معیار «شاخص معکوس» و «Grep» کم تراکم هستند، و «طبقه‌بندی» و «رده‌بندی هیستوگرام» به نقشه فشرده می‌شوند. بنابراین، ما ۱۲ شغل را در مجموعه D با ۸ شغل از معیارهای مبتنی بر نقشه و ۱۲ شغل در مجموعه E با ۸ شغل از معیارهای کاهش فشرده تنظیم کردیم.

جدول ۳,۲: مجموعه D: 12 کار مختلط با نقشه فشرده

Benchmark	Job #	Dataset	Map #	Reduce #
Classification	2	8GB Movie Rating Data	270	250
Classification	2	4GB Movie Rating Data	129	120
Wordcount	1	8GB Wikipedia	256	200
Wordcount	1	4GB Wikipedia	128	100
Histogram_ratings	2	8GB Movie Rating Data	270	250
Histogram_ratings	2	4GB Movie Rating Data	129	120
Histogram_movies	1	8GB Movie Rating Data	270	200
Histogram_movies	1	4GB Movie Rating Data	129	100

جدول ۳,۳: مجموعه E: 12 کار مختلط کم تراکم

Benchmark	Job #	Dataset	Map #	Reduce #
Invertedindex	2	8GB Wikipedia	256	250
Invertedindex	2	4GB Wikipedia	128	120
Wordcount	1	8GB Wikipedia	256	200
Wordcount	1	4GB Wikipedia	128	100
Grep[a-g][a-z]*	2	8GB Wikipedia	270	250
Grep[a-g][a-z]*	2	4GB Wikipedia	128	100
Histogram_movies	1	8GB Movie Rating Data	270	200
Histogram_movies	1	4GB Movie Rating Data	129	100

اول، شکل ۳,۵ نشان می دهد که طول و انصاف کلی با مجموعه A و مقادیر مختلف k. برای ساخت و ساز، FRESH-dynamic همیشه برتر از زمان‌بندی منصفانه با تنظیمات اسلات استاتیک است. بهترین عملکرد در زمان بندی زمانبندی منصفانه زمانی حاصل می شود که نسبت اسلات های نقشه به کاهش اسلات ۲:۲ (عادلانه-۲:۲) باشد. در مقایسه با Fair-2:2، FRESH-dynamic هنوز هم زمانی که $k = 5$ است، ٪۲۷,۶۲ طول عمر را بهبود می بخشد. در مقایسه با عملکرد عادلانه کلی با حجم کاری ساده، زمانبندی Fair با بارهای کاری ترکیبی بسیار بدتر عمل می کند (بهترین مقدار: حدود ۰,۸) زیرا مشاغل متنوع مراحل نقشه خود را در مقاطع زمانی مختلف به پایان می‌رسانند. از سوی دیگر، FRESH-Dynamic به طور قابل توجهی انصاف کلی را بهبود می بخشد، به خصوص زمانی که $k \geq 5$ (با مقادیر انصاف در حدود ۰,۹۵). با

این حال، زمان‌بندی معمولی Fair مقادیر انصاف بسیار کمتری را با بارهای کاری مختلط به دست می‌دهد (بهترین مقدار حدود ۰٫۸ است). انصاف کلی را با مقادیر مختلف k کنترل کنید. وقتی k بزرگتر از ۱ باشد، FRESH نسبت به زمانبندی منصفانه به انصاف بسیار بهتری دست می یابد.

شکل ۳٫۶ اجرای وظایف و تکالیف اسلات را نشان می دهد. رنگ های مختلف نشان دهنده مشاغل مختلف در مجموعه است. محور y شاخص شکاف را نشان می‌دهد، جایی که شکاف ۱~۴۰ همه اسلات‌های نقشه ممکن و شکاف ۴۱ ~ ۸۰ همه شکاف‌های کاهش ممکن در خوشه هستند. به یاد بیاورید که در کل ۴۰ اسلات در خوشه وجود دارد. کل فرآیند اجرا را می توان به سه مرحله تقسیم کرد. مرحله اول از ابتدا تا حدود ۶۵۰ ثانیه است. از آنجایی که هیچ کاری فاز نقشه خود را تمام نمی کند و هیچ کار کاهشی در دسترس نیست، تمام اسلات ها به فاز نقشه اختصاص داده می شوند. مرحله دوم از ۶۵۰ ثانیه تا ۱۵۰۰ ثانیه است. پس از اتمام مرحله نقشه‌برداری اولین کار، FRESH-dynamic شروع به استفاده از تعداد اسلات‌های نقشه و کاهش اسلات‌ها در خوشه با توجه به کل بارهای کاری باقی‌مانده در فاز نقشه و کاهش فاز می‌کند. در ابتدای این مرحله (حدود ۶۵۰ ثانیه)، تنها چند اسلات به عنوان اسلات کاهش اختصاص داده می شود، در حالی که اکثر اسلات ها هنوز توسط وظایف نقشه اشغال می شوند زیرا کل حجم کار باقی مانده در فاز نقشه سنگین تر است. از حدود ۸۰۰ ثانیه تا ۱۰۰۰ ثانیه، تعداد اسلات های نقشه مشابه تعداد اسلات های کاهش می شود، زیرا بارهای کاری باقی مانده در فاز نقشه و فاز کاهشی نزدیک می شوند. پس از ۱۰۰۰ ثانیه، حجم کار باقی مانده در نهایت غالب می شود و اسلات های بیشتری برای کاهش فاز اختصاص داده می شود. مرحله سوم، ۳۰۰ ثانیه آخر اجرا است، که در آن تمام وظایف نقشه به پایان می رسد و تمام اسلات ها به اسلات کاهش می یابند. علاوه بر این، شکل ۳٫۷ انصاف کلی را در طول آزمایش با مجموعه A نشان می دهد.

علاوه بر این، ما نتایج را با مجموعه B~E ارائه می کنیم. ما FRESH-Dynamic را با سه تنظیم مختلف k آزمایش می کنیم: فقط یک کار، نیمی از کارها، و همه کارها که به ترتیب با FRESH:1، FRESH:half و FRESH:all نشان داده می شوند. ما با Hadoop معمولی با زمان‌بندی FIFO، زمان‌بندی ظرفیت [۲۸] و زمان‌بندی منصفانه مقایسه می‌کنیم.

جدول ۳٫۴: مجموعه B: 6 کار مختلط

JobID	Benchmark	Dataset	Map #	Reduce #
01	Classification	8GB Movie Rating Data	270	250
02	Invertedindex	8GB Wikipedia	256	250
03	Wordcount	8GB Wikipedia	256	200
04	Grep[a-g][a-z]*	8GB Wikipedia	270	250
05	Histogram_ratings	8GB Movie Rating Data	270	250
06	Histogram_movies	8GB Movie Rating Data	270	200

جدول ۳٫۵: مجموعه C: 24 کار مختلط

Benchmark	Job #	Dataset	Map #	Reduce #
Classification	2	8GB Movie Rating Data	270	250
Classification	2	4GB Movie Rating Data	129	120
Invertedindex	2	8GB Wikipedia	256	250
Invertedindex	2	4GB Wikipedia	128	120
Wordcount	2	8GB Wikipedia	256	200
Wordcount	2	4GB Wikipedia	128	100
Grep[a-g][a-z]*	2	8GB Wikipedia	270	250
Grep[a-g][a-z]*	2	4GB Wikipedia	128	100
Histogram_ratings	2	8GB Movie Rating Data	270	250
Histogram_ratings	2	4GB Movie Rating Data	129	120
Histogram_movies	2	8GB Movie Rating Data	270	200
Histogram_movies	2	4GB Movie Rating Data	129	100

برای برنامه‌ریزی ظرفیت، دو صف ایجاد می‌کنیم و هر صف دارای همان تعداد اسلات وظیفه است. هر صف می تواند حداکثر ۹۰٪ اسلات در خوشه به دست آورد. همچنین، ما مشاغل را به طور مساوی به این صف ها جدا می کنیم. نتایج آزمون در شکل ۳،۸ و شکل ۹.۳ نشان داده شده است. در شکل ۳،۸، در بیشتر موارد، زمان‌بندی منصفانه بدترین عملکرد را با مجموعه‌های مختلف کار انجام می‌دهد. به طور متوسط، FRESH ۳۱،۳۲٪ از زمان ساخت را در مقایسه با زمانبندی Fair و ۲۵،۱٪ را به زمانبندی ظرفیت بهبود می بخشد. وقتی مجموعه‌ای از کارها نه فشرده و نه فشرده‌تر هستند (مجموعه B و مجموعه C)، FIFO به خوبی FRESH عمل می‌کند. با این حال، زمانی که حجم کار در نقشه و فازهای کاهشی نامتعادل است (مجموعه D و مجموعه E)، FRESH 24.47٪ از زمان ساخت را در مقایسه با FIFO بهبود می بخشد. به طور کلی، FRESH با مجموعه کارهای مختلف به یک عملکرد عالی و پایدار در طول تولید می‌رسد. در شکل ۳،۹، FIFO ظاهراً بدترین عملکرد را در انصاف دارد. وقتی تعداد کارهای در حال اجرا همزمان k بیشتر از ۱ باشد، عملکرد FRESH از زمانبندی Fair با مقادیر عادلانه بالای ۰،۹۵ در همه موارد آزمایش شده بهتر عمل می کند. به خصوص، در آزمایش با مجموعه B، FRESH تقریباً به ۱ انصاف دست می یابد. در نهایت، مجموعه B را روی یک خوشه بزرگ با ۲۰ گره اسلیو آزمایش می کنیم و نتایج در شکل ۳،۱۰ نشان داده شده است. ما می‌توانیم افزایش عملکرد ثابتی را از FRESH مانند خوشه کوچکتر ۱۰ گره اسلیو مشاهده کنیم. FRESH در مقایسه با زمان‌بندی منصفانه، ۳۱،۱٪ طول زمان ساخت را کاهش می‌دهد. به طور خلاصه، FRESH در هر دو بار کاری ساده و مختلط، بهبود قابل توجهی در ساخت و انصاف ایجاد می کند.

کارهای مرتبط

برنامه ریزی شغلی یک جهت مهم برای بهبود عملکرد یک سیستم Hadoop است. زمان‌بندی پیش‌فرض FIFO نمی‌تواند به طور منصفانه در یک خوشه مشترک با چندین کاربر و مشاغل مختلف کار کند. زمانبندی منصفانه [۲۹] و زمانبندی ظرفیت [۲۸] به طور گسترده ای استفاده می شود تا اطمینان حاصل شود که هر شغل می تواند سهم مناسبی از منابع موجود را به دست آورد. هر دوی آنها انصاف را به طور جداگانه در مرحله نقشه و مرحله کاهش را در نظر می گیرند، اما اجرای کلی کارها را نه.

برای بهبود عملکرد، Quincy [17] و Delay Scheduling [15] موقعیت داده ها را در مورد زمانبندی منصفانه بهینه می کنند. اما این تکنیک‌ها انصاف را با موقعیت داده‌ها عوض می‌کنند. Coupling Scheduler در [۳۰, ۳۱, ۳۲] با هدف کاهش قحطی شکاف‌های کاهش در زمان‌بندی منصفانه و تحلیل عملکرد با مدل‌سازی است.

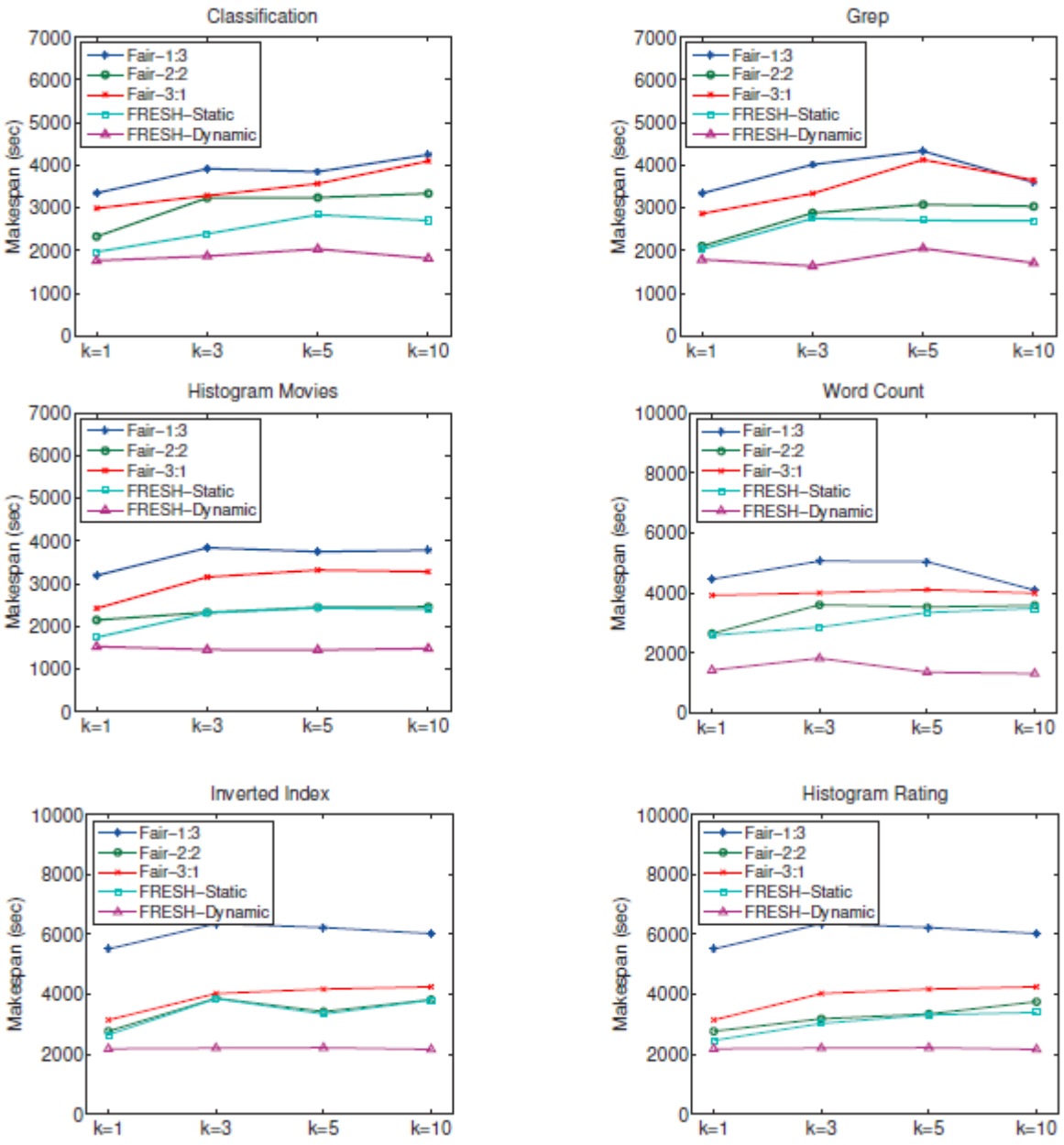
ویژگی های برنامه ریزی اساسی برای MapReduce. W. Wang [33] یک معماری صف جدید ارائه می‌کند و یک برنامه زمان‌بندی کار نقشه را پیشنهاد می‌کند تا تعادل مناسب بین داده-محلی و تعادل بار ایجاد شود. دسته دیگری از زمان‌بندی‌ها اهداف سطح کاربر را در حین بهبود عملکرد در نظر می‌گیرند. ARIA [18] مقادیر مناسبی از منابع را به مشاغل تخصیص می دهد تا مهلت از پیش تعریف شده را رعایت کند. iShuffle [34] فاز shuffle را از کاهش وظایف جدا می کند و یک سرویس پلت فرم برای مدیریت و زمان بندی خروجی داده از فاز نقشه ارائه می دهد. با این حال، همه این تکنیک ها هنوز بر اساس تنظیمات اسلات استاتیک هستند.

یکی دیگر از جهت‌گیری‌های مهم برای بهبود عملکرد در Hadoop، زمان‌بندی آگاهانه از منابع است. هدف RAS [22] بهبود استفاده از منابع در سراسر ماشین‌ها و رسیدن به مهلت تکمیل کار است. MROrchestrator [23] رویکردی را برای شناسایی استفاده از منبع وظیفه در هر TaskTracker به عنوان یک مدیر منبع محلی و تخصیص منابع به وظایف در JobTracker به عنوان یک مدیر منبع جهانی معرفی می کند. علاوه بر این، برخی از کارهای دیگر بر روی محیط‌های ناهمگن متمرکز شده‌اند. م. زهاریا و همکارانش. یک زمانبندی LATE [35] را برای متوقف کردن اجرای حدس و گمان غیرضروری در یک خوشه Hadoop ناهمگن پیشنهاد می کند. LsPS [36] از الگوهای اندازه کار ناهمگن فعلی برای تنظیم طرح‌های زمان‌بندی استفاده می‌کند.

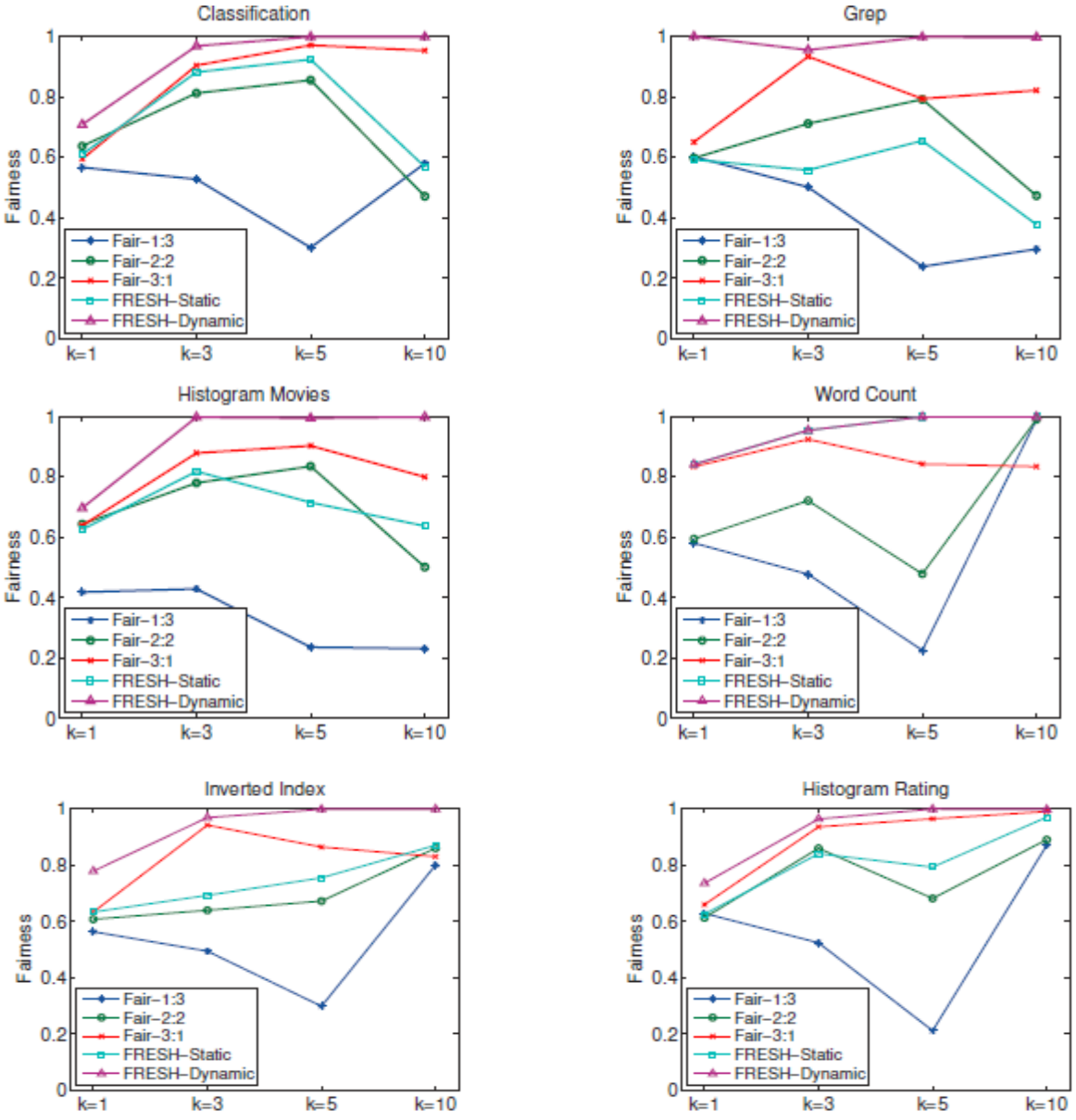
در نهایت، ما TuMM [37,38] را در کار قبلی خود پیشنهاد کرده‌ایم که به صورت پویا تنظیمات اسلات‌ها را در Hadoop بر اساس FIFO تنظیم می‌کند. با این حال، این کار، اجرای همزمان چندین کار را در نظر می‌گیرد که یک تنظیم مشکل کاملاً متفاوت و پیچیده‌تر است. ما همچنین هدف جدیدی از انصاف را در این کار گنجانده ایم.

خلاصه

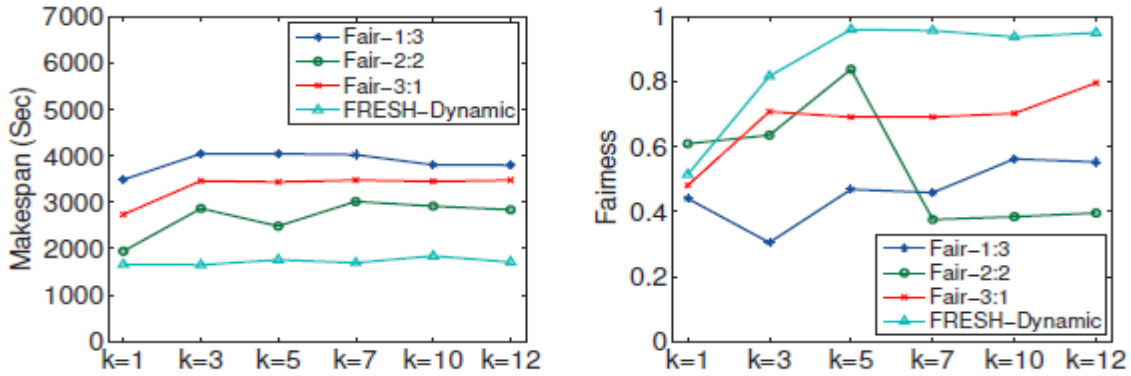
این کار بر روی مشکل تخصیص منابع به مراحل مختلف مشاغل متعدد تمرکز دارد. ما Hadoop MapReduce را به عنوان نماینده انتخاب می کنیم. ما یک خوشه Hadoop را مطالعه می کنیم که دسته ای از مشاغل MapReduce را ارائه می دهد. ما مشکلات پیکربندی اسلات و زمان‌بندی کار را هدف قرار می‌دهیم. ما FRESH، یک نسخه پیشرفته از Hadoop را توسعه می‌دهیم که از پیکربندی‌های اسلات استاتیک و پویا پشتیبانی می‌کند. علاوه بر این، ما یک تعریف جدید از عدالت کلی ارائه می دهیم. راه حل ما می تواند در حالی که انصاف را نیز به دست می آورد، makespan خوبی به همراه داشته باشد. ما آزمایش های گسترده ای را با حجم های کاری و تنظیمات مختلف انجام داده ایم. FRESH در مقایسه با یک سیستم Hadoop معمولی، بهبود قابل توجهی را در هر دو جنبه ساخت و انصاف نشان می دهد.



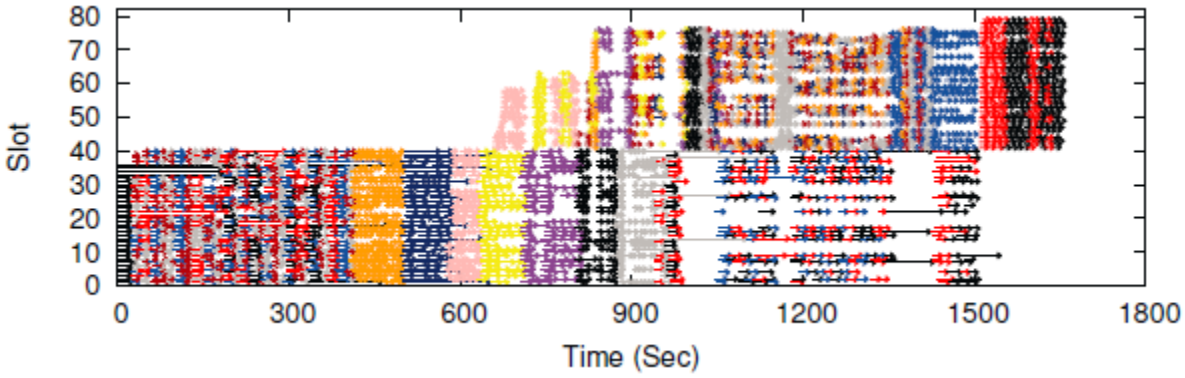
شکل ۳,۳: ایجاد حجم کار ساده تحت زمانبندی FRESH و Fair



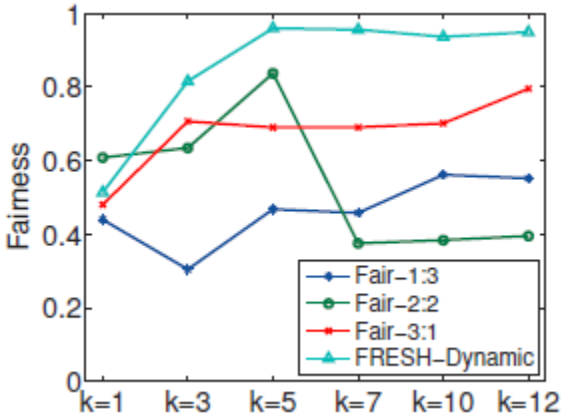
شکل ۳,۴: عادلانه بودن بارهای کاری ساده تحت زمانبندی FRESH و Fair



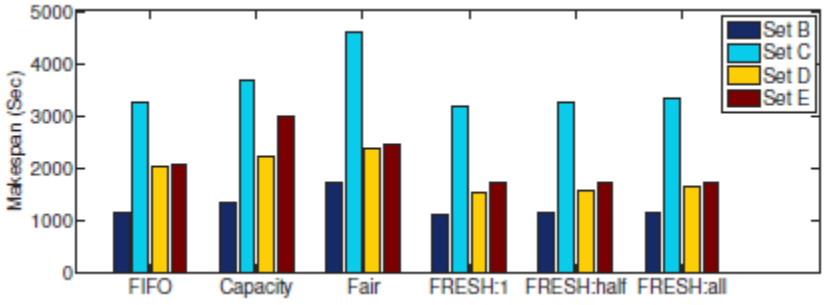
شکل ۳,۵: گستره و انصاف مجموعه A با مقادیر مختلف k



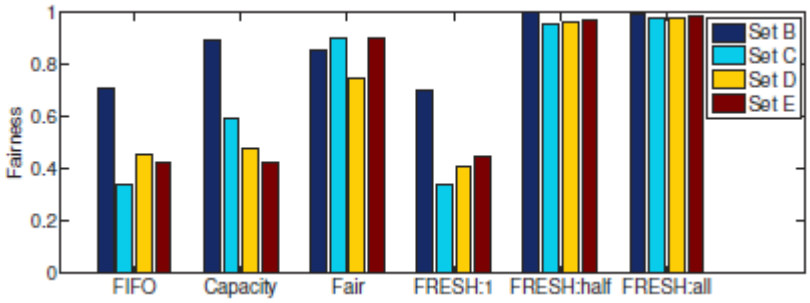
شکل ۳,۶: تنظیم اجرای وظایف و تکالیف اسلات با FRESH-dynamic (k = 5)



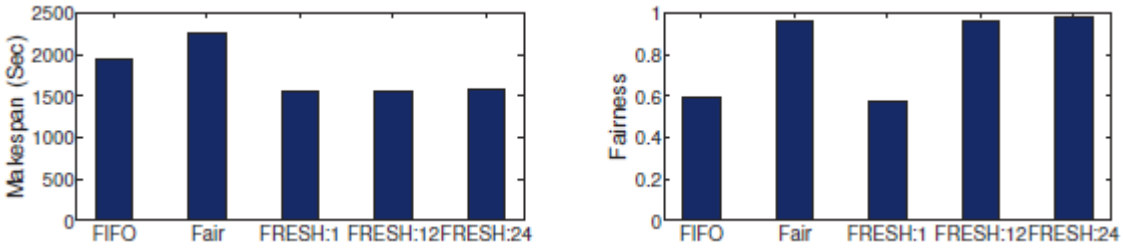
شکل ۳,۷: انصاف مجموعه A با مقادیر مختلف k



شکل ۳,۸: گستردگی مجموعه B~E (با ۱۰ گره اسلیو)



شکل ۳,۹: انصاف مجموعه B~E (با ۱۰ گره اسلیو)



شکل ۳,۱۰: گستردگی و انصاف مجموعه B با ۲۰ گره اسلیو

لیست منابع

[1] Idc reports. https://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf.

[2] Apache Hadoop. http://hadoop.apache.org.

[3] Apache hadoop nextgen mapreduce (yarn). http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[4] Benjamin Hindman, Andy Konwinski, Matei Zaharia, et al. Mesos: A platform for fine-grained resource sharing in the data center. NSDI, pages 22–22. USENIX, 2011.

[5] Bikas Saha, Hitesh Shah, Siddharth Seth, Gopal Vijayaraghavan, Arun Murthy, and Carlo Curino. Apache tez: A unifying framework for modeling and building data processing applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1357–1369, New York, NY, USA, 2015. ACM.

[6] Apache Spark. http://spark.apache.org.

[7] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, August 2009.

[8] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1099–1110, New York, NY, USA, 2008. ACM.

[9] Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: Sql and rich analytics at scale. In *Proceedings of the 2013ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 13–24, New York, NY, USA, 2013. ACM.

[10] A Storm is coming: more details and plans for release. https://blog.twitter. com/2011/a-storm-is-coming-more-details-and-plans-for-release.

[11] Apache Mahout. http://mahout.apache.org/.

[12] Jiayin Wang, Yi Yao, Ying Mao, Bo Sheng, and Ningfang Mi. Fresh: Fair and efficient slot configuration and scheduling for hadoop clusters. In *CLOUD*, 2014.

[13] Jiayin Wang, Yi Yao, Ying Mao, Bo Sheng, and Ningfang Mi. Optimize mapreduce overlap with a good start(reduce) and a good finish(map). In *IPCCC*, Dec 2015.

[14] Jiayin Wang, Teng Wang, Zhengyu Yang, Ningfang Mi, and Sheng Bo. eSplash: Efficient Speculation in Large Scale Heterogeneous Computing Systems. In *35th IEEE International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2016.

[15] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, et al. Delay scheduling:
A simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys*, pages 265–278, 2010.

[16] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance. In *MASCOTS*, Aug 2012.

[17] Michael Isard, Vijayan Prabhakaran, Jon Currey, et al. Quincy: Fair scheduling for distributed computing clusters. In *SOSP*, pages 261–276, 2009.

[18] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. Aria: Automatic resource inference and allocation for mapreduce environments. In *ICAC*, pages 235–244, 2011.

[19] Jorda Polo, David Carrera, Yolanda Becerra, et al. Performance-driven task co-scheduling for mapreduce environments. In *NOMS*, pages 373–380, 2010.

[20] Next generation mapreduce scheduler. http://goo.gl/GACMM".

[21] Xiao Wei Wang, Jie Zhang, Hua Ming Liao, and Li Zha. Dynamic split model of resource utilization in mapreduce. In *DataCloud-SC*, pages 21–30, 2011.

[22] Jord`a Polo, Claris Castillo, David Carrera, et al. Resource-aware adaptive scheduling for mapreduce clusters. In *Middleware*, 2011.

[23] B. Sharma, R. Prabhakar, S. Lim, M.T. Kandemir, and C.R. Das. Mrorchestrator: A fine-grained resource orchestration framework for mapreduce clusters. In *CLOUD*, pages 1–8, June 2012.

[24] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared systems. Technical report, Digital Equipment Corporation, December 1984.

[25] Amazon EC2 Instances. http://aws.amazon.com/ec2/instance-types/.

[26] Purdue mapreduce benchmarks suite. http://web.ics.purdue.edu/~fahmad/benchmarks.htm.

[27] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[28] Capacity scheduler. http://hadoop.apache.org/common/docs/r1.0.0/ apacity_scheduler.html.

[29] Fair scheduler. http://hadoop.apache.org/common/docs/r1.0.0/fair_scheduler.html.

[30] Jian Tan, Xiaoqiao Meng, and Li Zhang. Performance analysis of coupling scheduler for mapreduce/hadoop. In *INFOCOM*, pages 2586–2590, March 2012.

[31] Jian Tan, Xiaoqiao Meng, and Li Zhang. Delay tails in mapreduce scheduling. In *SIGMETRICS*, pages 5–16, 2012.

[32] Jian Tan, Xiaoqiao Meng, and Li Zhang. Coupling task progress for mapreduce resource-aware scheduling. In *INFOCOM*, pages 1618–1626, April 2013.

[33] Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang. Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality. In *INFOCOM*, pages 1609–1617, 2013.

[34] Yanfei Guo, Jia Rao, and Xiaobo Zhou. ishuffle: Improving hadoop performance with shuffle-on-write. In *ICAC*, pages 107–117, 2013.

[35] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, et al. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI, pages 29–42, 2008.

[36] Yi Yao, Jianzhe Tai, Bo Sheng, and Ningfang Mi. Scheduling heterogeneous mapreduce jobs for efficiency improvement in enterprise clusters. In *IM*, pages 872–875.

[37] Yi Yao, Jiayin Wang, Bo Sheng, et al. Using a tunable knob for reducing makespan of mapreduce jobs in a hadoop cluster. In *CLOUD*, pages 1–8, June 2013.

[38] Y. Yao, J. Wang, B. Sheng, C. Tan, and N. Mi. Self-adjusting slot configurations for homogeneous and heterogeneous hadoop clusters. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2015.

[39] Robert Ricci, Eric Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX*, 39(6), December 2014.

[40] S. Alspaugh Y. Chen and R. Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. VLDB, pages 1802–1813, 2012.

[41] Yi Yao, Jiayin Wang, Jason Sheng, Bo aMulti-resource Packing for Cluster Schedulersnd Lin, and Ningfang Mi. Haste: Hadoop yarn scheduling based on task-dependency and resource-

demand. In *Proceedings of the 2014 IEEE International Conference on Cloud Computing*, CLOUD ’14, pages 184–191, Washington, DC, USA, 2014. IEEE Computer Society.

[42] Y. Yao, H. Gao, J. Wang, N. Mi, and B. Sheng. Opera: Opportunistic and efficient resource allocation in hadoop yarn by harnessing idle resources. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, Aug 2016.

[43] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: Distributed, low latency scheduling. In *SOSP*, pages 69–84, 2013.

[44] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, et al. Omega: Flexible, scalable schedulers for large compute clusters. In *EuroSys*, pages 351–364, 2013.

[45] Michael Isard, Mihai Budiu, Yuan Yu, et al. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys*, pages 59–72, 2007.

[46] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[47] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: A flexible data processing tool. *Commun. ACM*, 53(1):72–77, January 2010.

[48] Tom White. Speculative execution. In *Hadoop: The Definitive Guide*, chapter 6. O’Reilly Media, Inc., 2012.

[49] B.Thirumala Rao, N.V.Sridevi, V.Krishna Reddy, and L.S.S.Reddy. Performance issues of heterogeneous hadoop clusters in cloud computing. 07 2012.

[50] Aysan Rasooli and Douglas G. Down. A hybrid scheduling approach for scalable heterogeneous hadoop systems. In *Proceedings of the 2012 SC Companion:High Performance Computing, Networking Storage and Analysis*, SCC ’12, pages 1284–1291, Washington, DC, USA, 2012. IEEE Computer Society.

[51] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, et al. Job scheduling for multi-user mapreduce clusters. Technical report, EECS Department, University of California, Berkeley, Apr 2009.

[52] Aysan Rasooli Oskooei and Douglas G. Down. Coshh: A classification and optimization based scheduler for heterogeneous hadoop systems. *Future Generation Comp. Syst.*, 36:1–15, 2014.

[53] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, J. Majors, A. Manzanares, and Xiao Qin. Improving mapreduce performance through data placement in heterogeneous hadoop

clusters. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–9, April 2010.

[54] Shekhar Gupta, Christian Fritz, Bob Price, Roger Hoover, Johan Dekleer, and Cees Witteveen. Throughputscheduler: Learning to schedule on heterogeneous hadoop clusters. In *Proceedings of the 10th International Conference on AutonomicComputing (ICAC 13)*, pages 159–165, San Jose, CA, 2013. USENIX.

[55] Xiaoqi Ren, Ganesh Ananthanarayanan, Adam Wierman, and Minlan Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 379–392, New York, NY, USA, 2015. ACM.

[56] N. S. Islam, X. Lu, M. Wasi ur Rahman, D. Shankar, and D. K. Panda. Tripleh: A hybrid approach to accelerate hdfs on hpc clusters with heterogeneous storage architecture. In *Cluster, Cloud and Grid Computing (CCGrid), 2015*

15th IEEE/ACM International Symposium on, pages 101–110, May 2015.

[57] Balaji Palanisamy, Aameek Singh, Ling Liu, and Bryan Langston. Cura: A cost-optimized model for mapreduce in a cloud. In *IPDPS*, 2013.