

به نام خدا

پروژه کارشناسی

مقدمه‌ای بر کلان داده‌ها

نام و نام خانوادگی دانشجو

سعید عطایی

شماره دانشجویی

۹۲۱۲۴۳۰۱۰۰

استاد پروژه

سرکار خانم دکتر انسان

استاد داور

جناب دکتر سوادی

بهمن ۱۳۹۵ – آذر ۱۳۹۶

چکیده :

بیگ دیتا یا کلان داده چند سالیست که در ادبیات فناوری اطلاعات به یک اصطلاح فراگیر تبدیل شده است. معمولاً، کلان داده‌ها به مجموعه داده‌هایی گفته می‌شود که توانایی دریافت، اکتساب، مدیریت و پردازش آن‌ها در یک زمان قابل قبول به وسیله فناوری اطلاعات و ابزارهای نرم‌افزاری و سخت‌افزاری سنتی وجود ندارد. عبارت **Big Data** مدت‌ها است که برای اشاره به حجم‌های عظیمی از داده‌ها که توسط سازمان‌های بزرگی مانند گوگل یا ناسا ذخیره و تحلیل می‌شوند مورد استفاده قرار می‌گیرد. اما به تازگی، این عبارت بیشتر برای اشاره به مجموعه‌های داده‌ای بزرگی استفاده می‌شود که رشد فزاینده‌ی میزان داده‌ها به حدی است که با ابزارهای مدیریتی و پایگاه‌های داده سنتی و معمولی قابل مدیریت نیستند. مشکلات اصلی در کار با این نوع داده‌ها مربوط به برداشت و جمع‌آوری، ذخیره‌سازی، جست‌وجو، اشتراک‌گذاری، تحلیل و نمایش آن‌ها است. این مبحث، به این دلیل هر روز جذابیت و مقبولیت بیشتری پیدا می‌کند که با استفاده از تحلیل حجم‌های بیشتری از داده‌ها، می‌توان تحلیل‌های بهتر و پیشرفته‌تری را برای مقاصد مختلف، از جمله مقاصد تجاری، پزشکی و امنیتی انجام داد و نتایج مناسب‌تری را دریافت کرد. تحقیقات در زمینه‌ی کلان داده‌ها باید روی چگونگی استخراج ارزش آن‌ها، چگونگی استفاده از داده‌ها و چگونگی تبدیل آن‌ها از گروهی از داده‌ها به کلان داده‌ها تمرکز کنند.

هدوپ یک فریم‌ورک متن‌باز برای ذخیره‌سازی امن و پردازش توزیع شده داده‌های حجیم می‌باشد که دارای دو بخش اصلی سیستم فایل توزیع شده هدوپ و موتور پردازش نگاشت و کاهش می‌باشد. ساختار هدوپ برای کار با داده‌های بزرگ طراحی شده است. لذا زمانی که ما دارای فایل‌های کوچک کلان هستیم، هدوپ نمی‌تواند برخورد مناسبی در مواجهه با این مساله از خود نشان دهد و باعث ایجاد بار سنگین بر روی گره اصلی هدوپ و افزایش زمان پردازش نگاشت و کاهش می‌شود. راهکارهای متفاوتی می‌تواند در برخورد با داده‌های کوچک به کار برده شود تا باعث بهبود عملکرد ذخیره‌سازی و پردازش و محاسبات هدوپ شود.

فهرست محتوا

فصل ۱.....	۱
کلان داده	۱
۱-۱ تعریف	۲
۱-۲ ویژگی ها	۳
۱-۳ انواع داده ها	۵
۱-۴ کاربرد ها	۶
۱-۵ چالش ها	۷
۱-۶ صنایعی که از کلان داده ها استفاده می کنند	۷
فصل ۲.....	۹
پایگاه داده‌های NoSQL	۹
۲-۱ مقدمه	۱۰
۲-۲ مدل‌های داده‌ای در پایگاه داده‌های غیر رابطه‌ای	۱۲
۲-۳ معیارهای گزینش پایگاه داده NoSQL	۱۳
فصل ۳.....	۱۶
سازگاری داده‌ها در پایگاه داده‌های غیر رابطه‌ای	۱۶
۳-۱ تئوری CAP	۱۷
۳-۲ ویژگی‌های ACID و BASE	۱۹
فصل ۴.....	۲۴
ماشین مجازی	۲۴
۴-۱ معرفی	۲۵
۴-۲ آموزش نصب	۲۶
فصل ۵.....	۳۰
داکر- DOCKER	۳۰
۵-۱ معرفی	۳۱
۵-۲ تفاوت آن با ماشین مجازی	۳۲

۳-۵	داکر برای چه کسانی مناسب است؟	۳۲
۴-۵	مکانیزم کاری DOCKER چگونه است؟	۳۳
۵-۵	داکر هاب	۳۳
۶-۵	کانتینر داکر	۳۳
۷-۵	نصب داکر بر روی اوبونتو	۳۴
۸-۵	استفاده از داکر برای پروژه	۳۶
۶	فصل	۳۸
	آپاچی هدوپ APACHE HADOOP	۳۸
۱-۶	معرفی	۳۹
۲-۶	چرا هدوپ؟	۴۰
۳-۶	معماری هدوپ	۴۰
۴-۶	سیستم فایل توزیع شده هدوپ (HDFS)	۴۱
۵-۶	موتور پردازش نگاشت/کاهش	۴۲
۶-۶	سرویس مدیریت منابع Yarn	۴۲
۷-۶	سرویس‌های هدوپ	۴۳
۸-۶	مراحل پردازش داده‌ها توسط هدوپ	۴۳
۷	فصل	۴۴
	طریقه نصب هدوپ در اوبونتو	۴۴
۱-۷	مراحل نصب	۴۵
۲-۷	اجرای مثال Word Count با هدوپ	۵۰
۳-۷	استفاده از ماشین مجازی هدوپ	۵۴
۸	فصل	۵۸
	اکوسیستم هدوپ	۵۸
۱-۸	سیستم فایل توزیع شده هدوپ - HDFS	۵۹
۲-۸	چهارچوب مدیریت منابع YARN	۶۰
۳-۸	موتور پردازش MapReduce	۶۲

۶۲APACHE MAHOUT ۸-۴ آپاچی ماہوت
۶۴APACHE ZOOKEEPER ۸-۵ آپاچی زو کیپر -
۶۵فصل ۹
۶۵Apache Hive آپاچی ہایو
۶۶۹-۱ معرفی
۶۷۹-۲ نصب آپاچی Hive بر روی اوبونتو
۶۹فصل ۱۰
۶۹APACHE SPARK آپاچی اسپارک
۷۰۱۰-۱ معرفی
۷۱۱۰-۲ نصب آپاچی اسپارک بر روی اوبونتو
۷۳۱۰-۳ اجرای مثال WordCount
۷۵فصل ۱۱
۷۵APACHE HBASE آپاچی اچ بیس
۷۶۱۱-۱ معرفی
۷۶۱۱-۲ نصب آپاچی hbase بر روی اوبونتو
۷۹منابع و مآخذ

فهرست شکل ها

۲.....	شکل ۱-۱.....
۳.....	شکل ۱-۲.....
۴.....	شکل ۱-۳.....
۴.....	شکل ۱-۴.....
۱۲.....	شکل ۲-۱.....
۱۸.....	شکل ۳-۱.....
۱۸.....	شکل ۳-۲.....
۲۲.....	شکل ۳-۳.....
۲۵.....	شکل ۴-۱.....
۲۶.....	شکل ۴-۲.....
۲۷.....	شکل ۴-۳.....
۲۷.....	شکل ۴-۴.....
۲۸.....	شکل ۴-۵.....
۲۸.....	شکل ۴-۶.....
۲۹.....	شکل ۴-۷.....
۳۰.....	شکل ۵-۱.....
۳۲.....	شکل ۵-۲.....
۳۳.....	شکل ۵-۳.....
۳۶.....	شکل ۵-۴.....
۳۶.....	شکل ۵-۵.....
۳۷.....	شکل ۵-۶.....
۳۷.....	شکل ۵-۷.....
۴۱.....	شکل ۶-۱.....
۴۲.....	شکل ۶-۲.....
۴۳.....	شکل ۶-۳.....
۵۳.....	شکل ۷-۱.....
۵۵.....	شکل ۷-۲.....
۵۵.....	شکل ۷-۳.....
۵۷.....	شکل ۷-۴.....
۵۹.....	شکل ۸-۱.....
۶۰.....	شکل ۸-۲.....
۶۱.....	شکل ۸-۳.....

۶۲.....	شکل ۸-۴
۶۳.....	شکل ۸-۵
۶۴.....	شکل ۸-۶
۶۶.....	شکل ۹-۱
۷۰.....	شکل ۱۰-۱
۷۱.....	شکل ۱۰-۲
۷۱.....	شکل ۱۰-۳
۷۲.....	شکل ۱۰-۴
۷۳.....	شکل ۱۰-۵
۷۴.....	شکل ۱۰-۶
۷۵.....	شکل ۱۱-۱
۷۸.....	شکل ۱۱-۲
۷۸.....	شکل ۱۱-۳

فصل ۱

کلان داده

۱-۱ تعریف

عبارت Big Data مدت‌ها است که برای اشاره به حجم‌های عظیمی از داده‌ها که توسط سازمان‌های بزرگی مانند گوگل یا ناسا ذخیره و تحلیل می‌شوند مورد استفاده قرار می‌گیرد؛ اما به تازگی، این عبارت بیشتر برای اشاره به مجموعه‌های داده‌ای بزرگی استفاده می‌شود که به قدری بزرگ و حجیم هستند که با ابزارهای مدیریتی و پایگاه‌های داده سنتی و معمولی قابل مدیریت نیستند. مشکلات اصلی در کار با این نوع داده‌ها مربوط به برداشت و جمع‌آوری، ذخیره‌سازی، جست‌وجو، اشتراک‌گذاری، تحلیل و نمایش آن‌ها است.

این مبحث، به این دلیل هر روز جذابیت و مقبولیت بیشتری پیدا می‌کند که با استفاده از تحلیل حجم‌های بیشتری از داده‌ها، می‌توان تحلیل‌های بهتر و پیشرفته‌تری را برای مقاصد مختلف، از جمله مقاصد تجاری، پزشکی و امنیتی، انجام داد و نتایج مناسب‌تری را دریافت کرد. بیشتر تحلیل‌های مورد نیاز در پردازش داده‌های عظیم، توسط دانشمندان در علومى مانند هواشناسی، ژنتیک، شبیه‌سازی‌های پیچیده فیزیک، تحقیقات زیست‌شناسی و محیطی، جست‌وجوی اینترنت، تحلیل‌های اقتصادی و مالی و تجاری مورد استفاده قرار می‌گیرد. حجم داده‌های ذخیره‌شده در مجموعه‌های داده‌ای Big Data، عموماً به خاطر تولید و جمع‌آوری داده‌ها از مجموعه بزرگی از تجهیزات و ابزارهای مختلف مانند گوشی‌های موبایل، حسگرهای محیطی، لاگ نرم‌افزارهای مختلف، دوربین‌ها، میکروفون‌ها، دستگاه‌های تشخیص RFID، شبکه‌های حسگر بی‌سیم و غیره با سرعت خیره‌کننده‌ای در حال افزایش است.



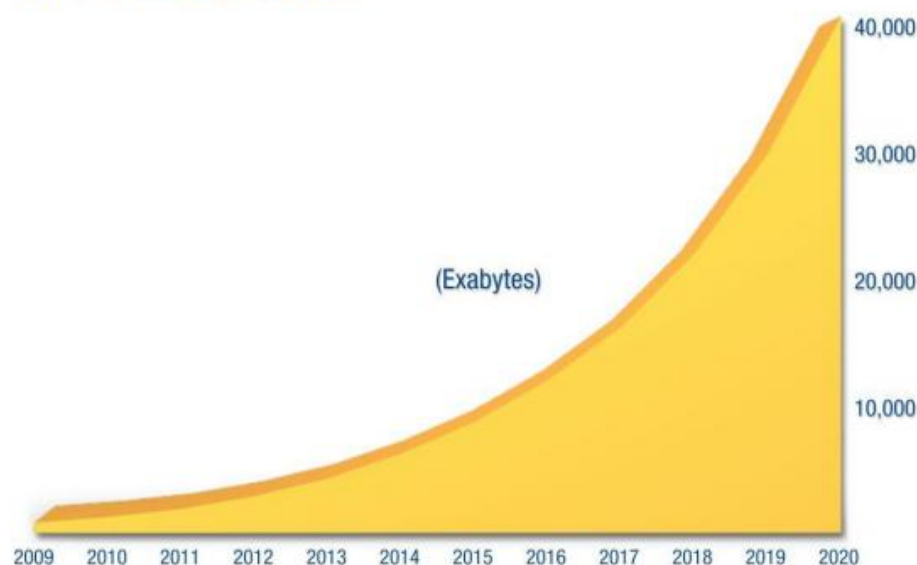
شکل ۱-۱

۱-۲ ویژگی ها

معمولا کلان داده را همراه با خصوصیات آن معرفی می کنند. پژوهشگران، سازمان ها و افراد فعال در حوزه کلان داده ویژگی های متفاوتی از کلان داده ارائه داده اند. برای مثال موسسه گارتنر سه ویژگی حجم، نرخ تولید و تنوع را به عنوان خصوصیات محیط کلان داده معرفی می کند. شرکت IBM علاوه بر این سه خصوصیت، صحت را نیز به عنوان یکی دیگر از خصوصیات محیط کلان داده معرفی کرده است. و یا در تعریفی دیگر علاوه بر خصوصیات بالا از ارزش نیز به عنوان یکی از ابعاد کلان داده یاد می شود. شرکت مایکروسافت هم ۶ ویژگی های حجم، تنوع، نرخ تولید، صحت، تغییر پذیری و قابل نمایان بودن داده ها را نیز به عنوان خصوصیات محیط کلان داده معرفی کرده است. هریک از این ویژگی ها بعد جدیدی از دنیای کلان داده را برای ما مشخص می کند. ما در اینجا در مورد پنج ویژگی حجم، نرخ تولید، تنوع، صحت و ارزش داده ها صحبت خواهیم کرد:

حجم: منظور از حجم "اندازه داده ها" می باشد، که با سرعت فزاینده ای روز به روز در حال گسترش است. اندازه ی داده هایی که توسط انسان ها، ماشین ها و تعامل آن ها در رسانه های اجتماعی تولید می شود، به تنهایی بسیار بزرگ است. پژوهشگران پیش بینی کرده اند که تا سال ۲۰۲۰، ۴۰ زتابایت (۴۰۰۰۰ اگزابایت) داده تولید خواهد شد، که نسبت به سال ۲۰۰۵، ۳۰۰ برابر افزایش خواهد داشت.

The Digital Universe: 50-fold Growth from the Beginning of 2010 to the End of 2020



Source: IDC's Digital Universe Study, sponsored by EMC, December 2012

شکل ۱-۲

نرخ تولید: منظور از نرخ تولید، میزان سرعتی است که منابع مختلف به‌طور روزانه، داده تولید می‌کنند. این حجم از جریان داده بسیار عظیم بوده و پیوسته در حال تولید است. در حال حاضر ۱۰۰۳ میلیارد “کاربرِ فعالِ روزانه” فیس‌بوک وجود دارند که از موبایل استفاده می‌کنند و سالانه ۲۲٪ نیز افزایش می‌یابند. این آمار نشان می‌دهد که تعداد کاربران رسانه‌های اجتماعی با چه سرعتی در حال افزایش است و روزانه داده‌ها با چه سرعتی تولید می‌شوند. چنان‌چه بتوانید این میزان از نرخ تولید را مدیریت کنید، به شناختی خواهید رسید که قادر خواهید بود بر پایه‌ی داده‌های زمان واقعی تصمیماتی اتخاذ کنید.

تنوع: از آن‌جایی که منابع فراوانی وجود دارند که می‌توانند به عنوان داده‌های اولیه در راه‌حل‌های کلان داده مورد تحلیل قرار بگیرند، قاعدتاً نوع داده‌هایی که ایجاد می‌شود نیز متفاوت است. این داده‌ها می‌توانند ساختاریافته، نیمه ساخت‌یافته و یا بدون ساختار باشند. از این‌رو، با تنوع داده‌های متعددی مواجه هستیم که روزانه تولید می‌گردند. پیش از این، معمولاً اکثریت داده‌ها در قالب‌های ساختاریافته وجود داشتند، اما همان‌طور که در تصویر پایین نشان داده شده است، امروزه داده‌هایی به شکل تصاویر، فایل‌های صوتی، ویدئو، داده‌های حس‌گر و غیره وجود دارند. به همین سبب، این تنوع داده، چالش‌های جدیدی در دریافت، ثبت، ذخیره‌سازی، تجزیه و تحلیل داده‌ها بوجود آورده است.



شکل ۱-۳

صحت: منظور از صحت، داده‌هایی است که به دلیل ناسازگاری و عدم یکپارچگی در میان آن‌ها، موجب شک و تردیدی در داده‌های موجود می‌شود. در تصویر پایین، مشاهده می‌کنید که مقادیر اندکی در این جدول از دست رفته است. هم‌چنین پذیرفتن بعضی از مقدارها هم سخت است، برای مثال – عدد ۱۵۰۰۰ در ردیف سوم، که حداقل مقدار را نشان می‌دهد، غیر ممکن به نظر می‌رسد. این ناسازگاری و غیر یکپارچگی‌ها، همان صحت داده‌ها می‌باشد.

Min	Max	Mean	SD
4.3	?	5.84	0.83
2.0	4.4	3.05	50000000
15000	7.9	1.20	0.43
0.1	2.5	?	0.76

شکل ۱-۴

گاهی اوقات ممکن است داده‌های موجود، نامرتب باشند و شاید اطمینان کردن به آن‌ها دشوار شود. با وجود اشکال مختلفی که کلان داده دارند، کنترل و نظارت بر کیفیت و صحت داده‌ها سخت می‌شود، مانند پست‌های توییتر که با هشتگ، مخفف، غلط‌های املائی و زبان محاوره نوشته می‌شوند. اغلب اوقات دلیل این عدم کیفیت و صحت داده‌ها، حجم بسیار انبوه داده‌ها است که از طریق راه‌های سنتی امکان تطابق و بررسی صحت آن‌ها وجود ندارد.

ارزش: پس از بررسی حجم، نرخ تولید، تنوع و صحت، به خصوصیت ارزش در تعریف کلان داده می‌پردازیم. دسترسی به داده‌های عظیم بسیار ارزشمند است، اما چنانچه نتوانیم این داده‌ها را ارزش‌گذاری کنیم، آن‌ها بلااستفاده باقی می‌مانند. حال اگر این داده‌ها به ارزشی که مد نظر ما است برسد، آیا برای سازمان‌هایی که این داده‌ها را تجزیه و تحلیل می‌کنند، مفید واقع خواهند شد؟ آیا سازمان‌هایی که بر روی کلان داده فعالیت می‌کنند، بازگشت سرمایه‌ی (ROI) چشم‌گیری خواهند داشت؟ در نهایت باید گفت چنانچه کار کردن بر روی کلان داده نتواند سودآوری لازم را داشته باشد، برای سازمان‌ها بی‌فایده خواهد بود.

۳-۱ انواع داده‌ها

داده‌های موجود در دنیای امروز را می‌توان به ۳ بخش تقسیم کرد:

داده‌های ساختاریافته: داده‌ها می‌توانند در فرمت ثابتی که “داده‌های ساختاریافته” نامیده می‌شوند، ذخیره و پردازش شوند. یک نمونه از داده‌های ساختاریافته، داده‌هایی هستند که در سیستم مدیریت پایگاه داده رابطه‌ای (RDBMS)، ذخیره می‌شوند. پردازش داده‌های ساختاریافته آسان است، چرا که این نوع داده‌ها دارای شمای ثابتی هستند. اغلب اوقات از زبان پرس و جوی SQL برای مدیریت این نوع داده‌ها استفاده می‌شود.

داده‌های نیمه ساختاریافته: داده‌های نیمه ساختاریافته، داده‌هایی هستند که ساختار رسمی “مدل داده” را ندارند، یعنی فاقد تعریف جدول در یک پایگاه داده رابطه‌ای هستند. با این وجود، این نوع داده‌ها از برخی ویژگی‌های سازمانی، همچون تگ‌ها و برخی نشان‌گذارهای دیگر که برای جدا کردن عناصر معنایی، که تجزیه و تحلیل داده‌ها را ساده‌تر می‌کند، بهره می‌برند. فرمت‌های داده XML و مستندات JSON دو نوع از متداول‌ترین داده‌های نیمه ساختاریافته هستند.

داده‌های بدون ساختار: داده‌هایی هستند که شکل و ساختاری مشخصی ندارند و به همین جهت RDBMS‌ها راه‌حل مناسبی برای ذخیره، تجزیه و تحلیل این داده‌ها نیستند. فایل‌های متنی و محتویات چندرسانه‌ای همچون تصاویر، فایل‌های صوتی و ویدئوها، نمونه‌هایی از داده‌های بدون ساختار هستند. سرعت رشد داده‌های بدون ساختار بیشتر از دیگر داده‌ها است.

و طبق نظر کارشناسان ۸۰٪ داده‌های یک سازمان، بدون ساختار هستند. پایگاه داده‌های غیر رابطه‌ای (NoSql) یکی از دسته ابزارهایی هستند که می‌توانند برای ذخیره و پردازش این نوع از داده‌ها بکار روند.

۱-۴ کاربردها

یکی از مهم‌ترین کاربردهای فناوری های Big Data، افزایش ضریب هوشمندی بنگاه‌های اقتصادی است. این فناوری‌ها بعد از تغذیه، گردآوری و پردازش داده‌ها، اقدام به تحلیل‌های هوشمند داده‌ها با هدف بهره‌برداری در هوش تجاری (هوشمندی کسب‌وکار) سازمان، داده‌کاوی و یادگیری ماشین، می‌کنند. این موارد می‌توانند سازمان‌ها، سیستم‌های اطلاعاتی و تصمیمات آن را با شرایط پیرامونی تطبیق‌پذیر (Adaptive) و پیشدستانه (Proactive) نماید.

امروزه تقریباً همه‌ی صنایع، از کاربردهای کلان داده به یک یا چند شیوه بهره می‌برند.

مراقبت‌های بهداشتی هوشمند: با تکیه بر داده‌های مربوط به بیماران، سازمان‌ها می‌توانند اطلاعات مفیدی را استخراج کند و اپلیکیشن‌هایی بسازند که می‌تواند پیشاپیش اوضاع بیمار را پیش‌بینی کند.

مخابرات: بخش‌های مخابراتی اطلاعات را جمع‌آوری و آن‌ها را تجزیه و تحلیل می‌کنند و راه‌حلهایی برای مشکلات مختلف ارائه می‌دهند. به عنوان یکی از کاربردهای کلان داده، شرکت‌های مخابراتی قادر هستند به‌طور چشم‌گیری، از دست رفتن بسته‌ها در سطح شبکه که در زمان‌های پر ترافیک در شبکه اتفاق می‌افتد را کاهش دهند، در نتیجه می‌توانند ارتباطی یک‌پارچه برای مشتری‌هایشان فراهم آورند.

خرده فروشی [آنلاین]: بازار خرده فروشی ارتباط تنگاتنگی با کلان داده دارد و یکی از بزرگ‌ترین ذینفع‌های کلان داده می‌باشد. زیبایی استفاده از کلان داده در خرده فروشی، درک رفتار مشتری است. موتور توصیه‌ی آمازون، پیشنهاداتی بر اساس تاریخچه‌ی مرورگر مشتری، ارائه می‌دهد.

کنترل ترافیک: تراکم ترافیک، چالشی بزرگ برای بسیاری از شهرها در سرتاسر جهان است. استفاده‌ی بهینه از داده‌ها و حس‌گرها، نکته‌ای کلیدی در کنترل هر چه بهتر ترافیک است، چراکه جمعیت شهرها به سرعت در حال افزایش است. سرعت پردازش داده‌ها، حجم تولید شده و همچنین تنوع داده‌ها بخوبی در این موردکاری قابل حس هستند.

کیفیت جستجو: هر زمان که اطلاعاتی را از گوگل استخراج می‌کنیم، هم‌زمان در حال ایجاد داده‌هایی برای گوگل هستیم. گوگل این داده‌ها را ذخیره می‌کند و از آن برای بهبود بخشیدن به کیفیت جستجو استفاده می‌کند.

۱-۵ چالش ها

کیفیت داده ها – بررسی کیفیت داده ها یکی از چالش های مهم در بحث کلان داده می باشد. مشکلی که در این مبحث مطرح می شود، به عامل “صحت” داده ها باز می گردد. معمولاً داده ها نامنظم، ناسازگار و غیر یکپارچه هستند و قبل از هرگونه تحلیل نیاز به مراحلی برای بررسی کیفیت داده دارند.

کشف – رسیدن به یک بینش و شناخت در زمینه ی کلان داده، مانند پیدا کردن سوزن در انبار کاه است. تجزیه و تحلیل داده ها در مقیاس پتابایت، با استفاده از الگوریتم های بسیار قدرتمند، به منظور یافتن الگوها و شناخت آن داده ها، بسیار دشوار است.

ذخیره سازی – هرچه مقدار داده ها در یک سازمان بیشتر باشد، متعاقباً مشکلاتی که برای مدیریت آن داده ها پیش می آید، پیچیده تر می شود. در راه حل های کلان داده یکی از سوالات اساسی این است که داده ها را با چه معماری ذخیره کنیم. در واقع به یک سیستم ذخیره سازی نیاز داریم که بتواند در صورت لزوم، بتوان به راحتی با تغییر مقیاس داده ها، این سیستم ذخیره ساز داده را نیز تغییر مقیاس داد.

امنیت – از آنجایی که اندازه ی داده ها عموماً بزرگ است، قاعده تاً حفظ امنیت آن هم چالش دیگری محسوب می شود. احراز هویت کاربران، محدود کردن دسترسی بر اساس کاربر، ثبت تاریخچه ی دسترسی به داده ها، استفاده ی صحیح از رمزگذاری داده ها و ... از جمله مواردی هستند که در حیطه ی امنیت جای می گیرند.

نبود استعداد کافی – پروژه های کلان داده ی بسیاری در سازمان های بزرگ وجود دارد، اما وجود یک تیم باتجربه از توسعه دهندگان، متخصصان علم داده و همچنین تحلیل گرانی که دانش کافی در زمینه ی داده داشته باشند، هنوز هم به صورت یک چالش باقی مانده است.

۱-۶ صنایعی که از کلان داده ها استفاده می کنند

بنگاه های اقتصادی و صناعی که به طور گسترده، از داده های عظیم (بیگ دیتا) استفاده می کنند عبارت اند از:

♦ صنعت بانکداری، بورس و اوراق بهادار

♦ صنعت بیمه

♦ تجارت خرده فروشی و عمده فروشی

♦ بنگاه های خدماتی

- ♦ ارائه‌دهندگان خدمات بهداشتی، درمانی و سلامت
- ♦ صنایع مخابرات، ارتباطات و اپراتورهای مخابراتی
- ♦ صنعت رسانه‌ها و سرگرمی
- ♦ بنگاه‌های آموزشی
- ♦ صنایع تولیدی و پخش
- ♦ دولت و خدمات دولتی
- ♦ صنعت حمل‌ونقل
- ♦ صنایع انرژی، آب، برق و گاز

معمولاً بنگاه‌های اقتصادی و سازمان‌های دولتی و خصوصی از کلان داده‌ها برای اهداف زیر استفاده می‌کنند:

- ♦ متمایز شدن از رقبا
- ♦ به دست آوردن سهم بازار بیشتر
- ♦ افزایش درآمد
- ♦ درک بهتر مشتریان
- ♦ سودآوری از طریق سرویس‌های جدید نوآورانه

بهره‌گیری از مزایای Big Data، معمولاً به سازمان‌ها، می‌تواند برای تحقق سه هدف حیاتی زیر مورد استفاده قرار گیرد:

- ♦ تحویل سرویس‌های هوشمندتری که منابع درآمدی جدیدی را تولید می‌کنند
- ♦ تحول در عملیات برای دستیابی به برتری تجاری و سرویس‌دهی
- ♦ ساخت زیرساخت‌های هوشمندتر برای هدایت و تقویت سازگاری و کیفیت تجربه مشتری

فصل ۲

پایگاه داده‌های NoSQL

پایگاه داده‌های رابطه‌ای و یا RDBMS از دهه‌ی ۱۹۷۰ مورد استفاده قرار گرفت و مطمئناً می‌توان از آن به عنوان یکی از فناوری‌های تکامل یافته برای ذخیره‌ی داده‌ها و روابطشان یاد کرد. با این حال، مشکلات ذخیره‌سازی در سیستم‌های مبتنی بر وب، محدودیت‌های پایگاه‌های داده رابطه‌ای را به چالش کشید. در نتیجه محققین و شرکت‌ها را وادار ساخت تا روی قالب‌های غیر سنتی ذخیره‌ی داده‌ها تحقیق و بررسی کنند. امروزه داده‌های کاربران می‌توانند تا مقیاس چند ترابایت در یک روز افزایش یابند و باید در کمترین زمان ممکن برای میلیون‌ها کاربر در سراسر دنیا قابل دسترس باشند.

تجزیه، تحلیل و به‌خصوص ذخیره‌سازی این حجم از اطلاعات در نوع خود یک چالش محسوب می‌شود. در چهارچوب یک سیستم تک‌گره، افزایش ظرفیت ذخیره‌سازیِ گره‌ی محاسباتی، به معنای افزایش حافظه‌ی اصلی و یا فضای دیسک بیشتر، با محدودیت‌های اساسی سخت‌افزار مواجه می‌شود. زمانی که یک گره با محدودیت ذخیره‌سازی مواجه می‌شود، هیچ جایگزینی برای آن وجود ندارد، اما در یک سیستم توزیع شده می‌توان داده‌ها را در بین گره‌های مختلف توزیع کرد. سابقاً، سیستم‌های RDBMS به گونه‌ای طراحی نشده بودند که بتوانند به راحتی توزیع شوند و در نتیجه افزودن گره‌های جدید برای متعادل کردن بار داده‌ها، امری بسیار پیچیده است. علاوه بر این، اغلب اوقات کارایی پایگاه داده‌ها به‌طور چشمگیری کاهش می‌یابد، چراکه عملیات‌های پیوند و تراکنش در محیط‌های توزیع شده، بسیار پرهزینه هستند. البته، این مسئله به این معنا نیست که پایگاه‌های داده‌ای RDBMS منسوخ شده‌اند، بلکه برای نوع نیازهای خاص دیگری طراحی شده‌اند و در صورتی که صحبت از مقیاس پذیری بیش از اندازه مطرح نباشد، می‌توان گفت همچنان می‌توانند به عنوان بهترین گزینه باشند.

اگر بخواهیم دقیق‌تر به پایگاه‌های داده‌ای NoSQL بپردازیم، باید گفت که به عنوان روش ذخیره‌سازی جایگزین و البته نه بر اساس مدل‌های رابطه‌ای، ارائه شده‌اند تا بتوانند مشکلاتی را که پیش‌تر به آن اشاره شد، برطرف سازند. اصطلاح “NoSQL” توسط کارلو استروزی در سال ۱۹۸۸ مطرح شد و از آن برای اشاره به پایگاه داده‌ای متن‌بازی به نام NoSQL استفاده می‌شد که فاقد رابط SQL است. بار دیگر این اصطلاح در سال ۲۰۰۹، توسط اریک اونز در چهارچوب پدیده‌ای پیرامون پایگاه‌های داده توزیع شده ظاهر گشت. همچنین از طرفی دیگر برخی از متخصصان علوم داده به این مسئله اشاره کرده‌اند که الگوهای مدیریت اطلاعات در حوزه‌های جدیدی مانند اینترنت اشیاء و یا شبکه‌های اجتماعی در زمینه‌ی ذخیره‌سازی داده‌ها، به تغییراتی بنیادی نیاز دارند. به همین دلیل، پایگاه‌های داده سنتی از عهده‌ی حجم کلان اطلاعاتی که در زمینه‌هایی همچون اطلاعات GPS، RFIDs، آدرس‌های IP، داده‌ها و فراداده‌های دستگاه‌های علمی و داده‌های حس‌گر تولید می‌شوند، بر نمی‌آیند و اصولاً مدل‌های ذخیره‌سازی آنها برای این موارد کاربرد، یک روش بهینه‌ای نیست.



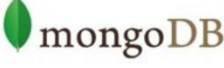












در حالت کلی، پایگاه داده‌های NoSQL غیر ساخت یافته هستند، یعنی دارای شمای ثابت نیستند و رابط استفاده از آن‌ها ساده است و به توسعه دهندگان این اجازه را می‌دهد تا بتوانند به سرعت شروع به استفاده از آن نمایند. علاوه بر این، به طور کلی این نوع پایگاه‌های داده مانع از عملیات پیوند در سطح ذخیره سازی داده‌ها می‌شوند، چراکه چنین عملیاتی اغلب پرهزینه هستند و این وظیفه را به لایه‌های دیگر واگذار می‌کنند. در واقع این توسعه دهنده است که باید تصمیم بگیرد که آیا عملیات پیوند در سطح برنامه انجام شود و یا به عنوان جاگزینی برای آن به غیر نرمال سازی داده‌ها بپردازد. در حالت اول، احتمالاً اتخاذ این تصمیم منجر به جمع‌آوری داده‌ها از گره‌های فیزیکی مختلف، بر اساس برخی از معیارها می‌شود و سپس بروی داده‌های جمع‌آوری شده عملیات پیوند انجام می‌شود. این رویکرد نیازمند به توسعه‌ی بیشتر در سطح برنامه است، در سال‌های اخیر چهارچوب‌های متعددی، از جمله مدل برنامه‌نویسی نگاشت/کاهش، اسپارک و یا ابزار Pregel به‌طور خارق‌العاده‌ای این وظیفه را به لطف فراهم نمودن مدلی برنامه‌نویسی برای پردازش‌های توزیع شده و موازی، آسان کرده‌اند. برای مثال مدل نگاشت/کاهش دارای دو فاز نگاشت و کاهش است که فاز نگاشت بروی داده‌ها که در بین نودهای فیزیکی پخش شده‌اند، اجرا شده و یکسری داده‌های میانی تولید می‌کند که در فاز کاهش عملیات تجمیع سازی و ایجاد نتایج از این داده‌های میانی انجام می‌پذیرد.

بسیاری از پایگاه‌های داده NoSQL برای حالت توزیع شده طراحی شده‌اند که به نوبه‌ی خود اجازه می‌دهند تا ظرفیتشان را، آن‌هم تنها با افزودن گره‌هایی به زیرساخت افزایش دهند؛ این قابلیت با نام “مقیاس پذیری افقی” هم شناخته می‌شود. در پایگاه‌های داده‌ای NoSQL (همانند بسیاری از سیستم‌های پایگاه داده‌ای توزیع شده)، اغلب از مکانیسم بخش بندی برای دستیابی به مقیاس پذیری افقی استفاده می‌شود، که شامل تقسیم کردن و تبدیل رکورد داده‌ها به بخش‌های مستقل و متعدد یا همان shard با استفاده از معیاری خاص نظیر شماره شناسه رکورد می‌شود. مکانیزم به‌کار گرفته شده دیگر مکانیسم تکرار است. یعنی رکورد داده‌ها در میان سرورهای متعدد تکرار می‌شود. این امر اجازه‌ی افزایش توان عملیاتی و رسیدن به قابلیت دسترس پذیری بالا را می‌دهد. هر دو مکانیسم بخش‌بندی و تکرار، دو مفهومی هستند که می‌توانند با ترکیب یکدیگر راه‌حلی برای مقیاس پذیری افقی در پایگاه داده‌های غیر رابطه‌ای بکار گرفته شوند.

در اکثر پیاده‌سازی‌ها، الزامات و نیازمندی‌های سخت‌افزاری هر گره مطابق با مشخصات سرورهای ارزان قیمت و معمولی است تا بتوان هزینه‌های ساخت چنین سیستم‌هایی را کاهش داد و هم‌چنین جایگزینی گره‌های معیوب راحت باشد.

۲-۲ مدل‌های داده‌ای در پایگاه داده‌های غیر رابطه‌ای

پایگاه داده‌های NoSQL به دسته‌بندی‌های مختلفی تقسیم می‌شوند، که هر کدام نوع خاصی از مدل داده را برای ذخیره‌سازی داده توصیف می‌کنند. این دسته‌بندی‌ها عبارتند از:

Document Database	Graph Databases
 Couchbase  MarkLogic  mongoDB	 Neo4j  TITAN  InfiniteGraph The Distributed Graph Database
Key-Value Databases	Wide Column Stores
 redis  amazon DynamoDB  AEROSPIKE  riak	 accumulo  HYPERTABLE  Cassandra  APACHE HBASE  Amazon SimpleDB

شکل ۲-۱

- ♦ **کلید/مقدار:** این نوع از پایگاه داده‌ها اجازه‌ی ذخیره کردن داده‌های دلخواه را تحت نظارت یک کلید می‌دهند. روش کار آن‌ها مشابه یک جدول هَش معمولی است، با این تفاوت که از کلیدهای توزیعی (و مقدارها) در میان مجموعه‌ای از گره‌های فیزیکی بهره می‌برد. از جمله متداول‌ترین محصولات در این بخش می‌توان به Redis، Riak، DynamoDB و BerkeleyDB اشاره کرد.
- ♦ **سطر گسترده یا مبتنی بر ستون:** در این مدل بجای ذخیره کردن سطری داده‌ها (یعنی همان شیوه‌ای که در پایگاه‌های داده‌ای رابطه‌ای کاربرد دارد)، داده‌ها در این نوع از پایگاه‌های داده ستونی ذخیره می‌شوند. بنابراین، بعضی از سطرها ممکن است شامل بخشی از ستون‌ها نباشند. این پایگاه داده‌ها امکان انعطاف‌پذیری در تعریف داده را می‌دهند و اجازه‌ی اعمال الگوریتم‌های فشرده‌سازی داده را در هر ستون می‌دهند. گذشته از این، ستون‌هایی که اغلب با یکدیگر جستجو نمی‌شوند، می‌توانند در گره‌های مختلف توزیع شوند. Cassandra، HBase و BigTable نمونه‌هایی از این مدل پایگاه داده‌ها محسوب می‌شوند.

♦ **سندگرا:** سند مجموعه‌ای از فیلدها به همراه مقادیر است. برای مثال: نام= “رضا”، نام خانوادگی= “بخشایشی” یک سند دو فیلدی می‌باشد. اکثر این نوع از پایگاه‌های داده، سندها را در قالب داده‌های نیمه ساخت یافته هم‌چون XML، JSON و یا BSON ذخیره می‌کنند. عملکرد آن‌ها مشابه پایگاه‌های داده کلید/مقدار است، با این تفاوت که در این مورد کلید همواره شناسه سند می‌باشد و مقدار هم سندی از پیش تعریف شده از انواع شناخته (مثل JSON یا XML) شده است که اجازه‌ی جستجو در فیلدها مختلف سند را می‌دهد. در این مدل پایگاه داده می‌توان به MongoDB، CouchDB اشاره کرد.

♦ **گرافی:** این نوع پایگاه‌های داده به قصد ذخیره کردن داده‌ها در ساختاری گراف مانند پدید آمده‌اند. داده‌ها از طریق رأسها و کمان‌ها و ویژگی‌های مخصوص خودشان ارائه می‌شوند. اغلب پایگاه‌های داده گرافی حتی زمانی که رأس‌ها در گره‌های فیزیکی جداگانه‌ای واقع شده باشند قادر به پیمایش گرافی کارآمدی هستند. علاوه بر این، اخیراً این نوع از پایگاه داده‌ای به خاطر کاربرد داده‌های اجتماعی، بسیار مورد توجه قرار گرفته است. این میزان از توجه، پیاده‌سازی‌های جدیدی را به ارمغان آورده تا با نیازهای امروزی مطابقت داشته باشد. از Neo4j، Titan و InfoGrid می‌توان به عنوان نمونه‌ای از پایگاه داده‌های گرافی یاد کرد.

۲-۳ معیارهای گزینش پایگاه داده NoSQL

برخی از معیارهایی که ممکن است در گزینش پایگاه داده‌ای NoSQL به عنوان سیستم‌های ذخیره‌سازی داده‌ها مفید واقع شوند، آورده شده است:

♦ **تجزیه و تحلیل داده‌ها:** برخی موقعیت‌ها ایجاب می‌کند که دانش و اطلاعاتی از داده‌های ذخیره شده در پایگاه داده استخراج کنیم. در میان روش‌های موجود برای اجرای وظایف در محیط کلان داده، مدل‌هایی نظیر نگاشت/کاهش و یا ابزار اسپارک در میان سایرین برجسته‌تر هستند. در بسیاری از این چهارچوب‌های پردازشی، توسعه دهنده باید کد جستجو را به زبان برنامه‌نویسی معینی ارائه کنند. اگرچه این روش نسبت به اعمال دستورات جستجوی GROUP BY، SELECT ... بسیار پیچیده‌تر است، ولی با همه‌ی این اوصاف برای محیط کلان داده که حجم و سرعت پردازش داده‌ها بسیار اهمیت دارد، مناسب‌تر هستند. زبان‌هایی هم‌چون Pig و Hive توسعه برنامه‌ها را با نگاشت/کاهش و یا چهارچوب SQL در اسپارک توسعه برنامه‌ها مبتنی بر درخواست‌های رابطه‌ای را آسان‌تر می‌کنند.

♦ **مقیاس‌پذیری:** پایگاه داده‌های NoSQL به منظور ذخیره‌سازی حجم عظیمی از داده‌ها و یا پشتیبانی از پردازش مورد نیاز، به واسطه‌ی افزودن گره‌های جدید به سیستم، طراحی شده‌اند. علاوه بر این، عموماً این سیستم‌ها بر

اساس این فلسفه طراحی شده‌اند که شکست و خطا در گره‌ها امری عادی است و همیشه نسخه‌های تکثیر شده‌ی داده‌ها باید آماده‌ی دریافت درخواست‌ها باشند. این نوع از طراحی در صورت خرابی در یکی از گره‌های خوشه، از استحکام خوبی برخوردار است. در برخی از موارد، زمانی که خرابی گره‌ها دائمی به نظر می‌رسند، داده‌ها مجدداً به‌طور خودکار در میان گره‌های در دسترس در کلاستر توزیع می‌گردند.

♦ **انعطاف‌پذیر در شمای داده‌ها:** پایگاه‌های داده NoSQL از شمای ثابتی برخوردار نیستند. پایگاه داده‌های کلید/مقدار هیچ‌گونه پیش‌فرضی درباره‌ی مقدارهای کلید ارائه نمی‌دهند. پایگاه داده‌های سندگرا و سطر گسترده پذیرای تعداد زیادی از ستون-مقدارها هستند. در پایگاه داده‌ای گرافی، رأس‌ها و کمان‌ها می‌توانند هر نوع ساختاری داشته باشند. در مقابل، پایگاه‌های داده‌ای رابطه‌ای از مجموعه جداولی با شما ثابت تشکیل شده است تعداد فیلدهای آن در هر سطر ثابت است.

♦ **استقرار سریع:** به‌طور کلی، سیستم‌های NoSQL می‌توانند به آسانی در یک خوشه مستقر شوند. پیکربندی تکرارها و بخش‌بندی اصولاً به‌طور خودکار انجام می‌گیرند و انتخاب آن‌ها را سرعت می‌بخشد.

♦ **آگاهی از مکان داده‌ها:** در مجموع از آنجایی که پایگاه‌های داده‌ای NoSQL مبتنی بر مدل‌های توزیع شده طراحی شده‌اند، دانستن قرارگیری مکان داده‌ها در خوشه و ارسال درخواست به گره‌ایی که داده در آن قرار دارد استفاده از پهنای باند شبکه را بطور چشمگیری بهبود می‌دهد. معمولاً این کار با استفاده از روش‌هایی نظیر مشخص کردن اینکه هر گره باید پذیرای چه محدوده‌ای از کلیدها باشد، صورت می‌پذیرد.

علاوه بر این، انتخاب پایگاه داده NoSQL باید بر اساس نوع داده‌ای که قرار است ذخیره شود و همچنین روش دسترسی به آن (خواندن و نوشتن) باشد. مثالی بدیهی برای آن، وب‌سایت‌هایی است که در هر ثانیه میلیون‌ها بازدید کننده دارند و داده‌هایشان برای پشتیبانی از عملکردی جدید می‌تواند کاملاً تغییر یابند. فیسبوک، گوگل و آمازون برخی از نمونه‌های بارز این مثال‌ها هستند. در این سیستم‌ها داده‌ها می‌تواند بدون هیچ‌گونه محدودیتی رشد یابند، در نتیجه زیرساخت سیستم باید اجازه‌ی افزایش فضای ذخیره‌سازی را، بدون آن‌که هیچ خللی در عملکرد آن وارد شود، بدهد. در این مواقع، به ندرت از پایگاه‌های داده‌ی رابطه‌ای استفاده می‌شود و فناوری‌های متعدد غیر رابطه‌ای از عهده‌ی مدیریت این نیازها برمی‌آید.

باید توجه داشته باشید که پایگاه‌های داده‌ای ترکیبی نیز وجود دارند که قادر به ذخیره کردن بیش از یک شمای داده هستند. برای نمونه می‌توان از OpenLink Virtuoso، OrientDB و AlchemyDB به عنوان پایگاه‌های داده‌ای چند-شمایی نام برد.

اخیراً سیستم پایگاه داده جدیدی به نام “پایگاه‌های داده NewSQL” به عنوان راه‌حلی در کنار پایگاه‌های داده NoSQL و پایگاه‌های داده سنتی RDBMS معرفی شده است. پایگاه‌های داده‌ای NewSQL پایگاه‌های داده‌ای رابطه‌ای هستند که بخش‌بندی، تکرار خودکار و پردازش تراکنش‌های توزیع شده را پشتیبانی می‌کنند و خاصیت ACID را حتی در سراسر بخش‌ها تضمین می‌کنند. برای ذکر نمونه از این نوع پایگاه داده می‌توان به Nuodb، VoltDB و Clustrix اشاره کرد. برای مثال Google Spanner سیستم پایگاه داده‌ای توزیعی است که توسط شرکت گوگل خلق شده است و تراکنش‌های توزیع شده را پشتیبانی می‌کند و به عنوان جایگزینی برای Megastore که سیستم ذخیره‌سازی بر اساس BigTable است، طراحی شده است. با این همه، همانند پایگاه‌های داده NoSQL، ضروری است که پایگاه‌های داده NewSQL قبل از آن که توسط سازمانی به کار گرفته شود، مورد تجزیه و تحلیل عمیق بگیرد.

سازگاری داده‌ها در پایگاه داده‌های غیررابطه‌ای

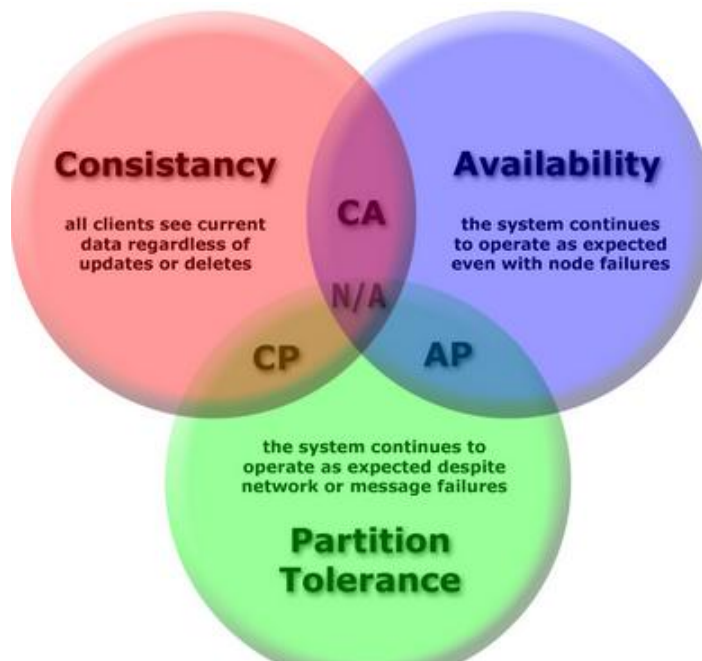
سازگاری و دسترس پذیر بودن داده‌ها در سیستم‌های توزیع شده از جمله مهم ترین چالش‌های طراحی یک چنین سیستم‌هایی می‌باشد. پایگاه داده‌های غیررابطه‌ای نیز نوعی از سیستم‌های توزیع شده محسوب می‌شوند و ایجاد سازگاری داده‌ها در بین گره‌های یک خوشه پایگاه داده، باعث شده است تا طراحی و نحوه تعامل با این پایگاه داده‌ها نسبت به پایگاه داده‌های رابطه‌ای متفاوت باشد.

۳-۱ تئوری CAP

در طراحی سیستم‌های توزیع شده تئوری وجود دارد که بیان می‌کند در یک سیستم توزیع شده باید از بین سه ویژگی سازگاری داده، در دسترس بودن و تحمل در برابر تقسیم خوشه دو مورد را انتخاب کرد. این محدودیت ذاتی سیستم‌های توزیع شده، که با نام تئوری CAP یا تئوری برور شناخته می‌شود، بیان می‌کند که غیرممکن است ویژگی‌های زیر را در یک سیستم توزیع شده به صورت همزمان داشته باشیم. از آنجایی که پایگاه داده‌های غیررابطه‌ای نیز نوعی سیستم توزیع شده بحساب می‌آیند، در نتیجه این تئوری در مورد پایگاه داده‌های غیررابطه‌ای نیز صدق می‌کند. به منظور درک بهتر این تئوری هریک از این ویژگی‌ها را تعریف می‌کنیم:

- ♦ **سازگاری:** اگر این ویژگی برقرار شود، تضمین می‌شود که در زمان خواندن داده‌ها از پایگاه داده بروزترین داده در اختیار کاربر قرار بگیرد و یا سیستم خطایی مبنی بر نقض سازگاری ارسال می‌کند.
- ♦ **در دسترس بودن:** برقرار بودن این ویژگی بیان می‌کند که سیستم در صورت وجود پاسخ حتماً آن را به عنوان پاسخ پرس و جو ارسال می‌کند. اگرچه ممکن است این پاسخ بروزترین داده در پایگاه داده نباشد.
- ♦ **تحمل در برابر تقسیم خوشه:** اگر سیستم دارای این ویژگی باشد، حتی با وجود شکست در برخی از قسمت‌های شبکه (و تقسیم شدن خوشه به چندین قسمت مجزا) بتواند به کار خود ادامه دهد.

در سال ۲۰۰۰ میلادی، اریک برور این فرضیه را مطرح کرد، که در هر زمان تنها دو مورد از سه مشخصه گفته شده قابل تضمین در یک سیستم توزیع شده است. چند سال بعد، گیلبرت و لینچ این فرضیه را به صورت رسمی و مدون بیان کردند، و نتیجه گرفتند که در هر سیستم توزیع شده تنها دستیابی به یکی از ترکیب‌های زیر ممکن است: AP (در دسترس بودن و تحمل در برابر تقسیم خوشه)، CP (سازگاری و تحمل در برابر تقسیم خوشه) یا AC (در دسترس بودن و سازگاری). جدول زیر، برخی از پایگاه داده‌های غیررابطه‌ای را براساس هریک از این سه دسته نشان می‌دهد.



شکل ۳-۱

در هر گروه، پایگاه داده‌ها براساس ویژگی‌های تئوری CAP، مجدداً طبقه‌بندی می‌شوند. همان‌طور که نشان داده شده است، بیشتر پایگاه داده‌های بررسی شده در گروه‌های «AP» و «CP» دسته‌بندی می‌شوند. زیرا فراهم نکردن P (تحمل در برابر تقسیم خوشه) در سیستم‌های توزیع شده تنها با این فرض ممکن است، که شبکه زیرساخت هیچ‌گاه دچار خرابی نشده و اتصال قطع نخواهد شد، که در عمل در سیستم‌های توزیع شده این امر با تقریب بسیار بالایی غیرممکن است. برخی از پایگاه داده‌ها، نظیر MongoDB، می‌توانند به گونه‌ای پیکربندی شوند که فراهم کردن کامل ویژگی سازگاری را با فدا کردن مقداری از دسترس پذیری و یا eventual consistency را با فراهم کردن دسترسی پذیری بالا تضمین کنند؛ به همین منظور در هر دو ستون AP و CP مشاهده می‌شوند.

الگوی داده	AP	CP	AC
کلید/مقدار	Riak, Infinispan, Redis, Voldemort, Hazelcast	Infinispan, Membase/CouchBase, BerkeleyDB	Infinispan
سطر گسترده	Cassandra	HBase, Hypertable	–
سندگرا	MongoDB, RavenDB, CouchDB, Terrastore	MongoDB	–
گرافی	Neo4J, HypergraphDB, BigData, AllegroGraph, InfoGrid, InfiniteGraph	InfiniteGraph	–

شکل ۳-۲

البته پس از چندین سال تئوری جدیدتری نسبت به تئوری CAP با عنوان تئوری PACELC مطرح شد که این تئوری عنوان می‌کند اگر سیستم دچار هیچگونه نقصی نشد (خوشه بصورت نرمال بکار خود ادامه دهد و هیچگونه شکستی آن را به دو یا چند قسمت تقسیم نکند) نیز باید بین تاخیر پاسخ و سازگاری داده‌ها یکی را انتخاب کنیم و در واقع موازنه‌ای بین این دو فاکتور برقرار کنیم.

۳-۲ ویژگی‌های BASE و ACID

در سال ۱۹۷۰ میلادی، جیم گری مفهوم تراکنش را که به معنی واحد کاری در یک سیستم پایگاه داده است، ارائه کرد. یک تراکنش باید چهار ویژگی اصلی را دارا باشد: تجزیه‌ناپذیری، سازگاری، ایزوله بودن و پایایی. این ویژگی‌ها، که با سرواژه ACID شناخته می‌شوند، طراحی سیستم‌های پایگاه داده را پیچیده می‌کنند؛ و این پیچیدگی در سیستم‌های پایگاه داده توزیع شده، که داده در بخش‌های مختلفی داخل یک شبکه کامپیوتری پخش می‌شوند، بیش‌تر خواهد بود. با این حال، این خصیصه، با تضمین اینکه هر عملیات، سازگاری در پایگاه داده را حفظ می‌کند، کار توسعه‌دهندگان را ساده می‌کند. همچنین به دلیل آنکه عملیات‌ها ذاتاً مستعد شکست و ایجاد تأخیر در شبکه هستند و به منظور تضمین موفقیت تراکنش، پیش‌بینی‌های فوق‌العاده‌ای باید در نظر گرفته شود.

در سال‌های اخیر، سیستم‌های مدیریت پایگاه‌داده‌های رابطه‌ای توزیع شده، به منظور حفظ سازگاری داده‌ها در طول پارتیشن‌ها، امکان اجرای تراکنش‌ها به وسیله پروتکل‌های خاص را فراهم کرده‌اند. نمونه‌ای از این RDBMSها، مگااستور است، پایگاه‌داده‌ای توزیع شده که قابلیت پشتیبانی از تراکنش‌های ACID در جداولی خاص و پشتیبانی محدود تراکنش‌ها در جداول داده مختلف را داراست.

مگااستور توسط BigTable پشتیبانی می‌شود؛ اما برخلاف BigTable، برای پشتیبانی از روابط سلسله‌ای در بین جداول، از یک زبان اسکیمای استفاده می‌کند و مدلی نیمه‌رابطه‌ای را فراهم می‌کند. با این‌که مگااستور (در مقایسه با استفاده مستقیم از BigTable) کارایی خوبی نداشت، در بسیاری از کاربردها، نیاز به سادگی اسکیمای تضمین همگام‌سازی رونوشت‌ها احساس می‌شد. مانند بسیاری از پایگاه‌های داده، رونوشت‌ها در مگااستور توسط نسخه‌ای از الگوریتم Paxos همگام می‌شوند. یکی از پروتکل‌های رایج برای اجرای تراکنش‌ها در محیط‌های توزیع شده در این سیستم‌ها، PC₂ (پروتکل تثبیت دو مرحله‌ای) است. این پروتکل حتی در سرویس‌های وب نیز رواج یافته، و استفاده از تراکنش‌ها را حتی در معماری‌های مبتنی بر REST نیز ممکن ساخته است. پروتکل PC₂، از دو بخش اصلی تشکیل شده است:

مرحله‌ای که مولفه هماهنگ‌کننده، از پایگاه‌داده‌هایی که درگیر تراکنش هست درخواست عملیات پیش‌تثبیت را می‌کند. اگر تمام پایگاه‌داده‌ها تمام عملیات‌ها را به اتمام برسانند، مرحله ۲ صورت می‌گیرد. در غیر این صورت، حتی اگر یکی از پایگاه‌داده‌ها تراکنش را نپذیرند (یا نتوانند پاسخ دهد)، تمامی پایگاه‌داده‌ها تغییرات صورت گرفته را بازگردانی می‌کنند. هماهنگ‌کننده از پایگاه‌داده‌ها درخواست اجرا عملیات تثبیت را می‌کند. اگر هر کدام از پایگاه‌داده‌ها عملیات تثبیت را رد کنند، بازگردانی در تمام پایگاه‌داده‌ها اجرا می‌شود.

براساس تئوری CAP، استفاده از پروتکلی مانند PC₂ (برای مثال در سیستم CP) اثرات منفی روی دسترس‌پذیری سیستم خواهد داشت. به این معنی که اگر پایگاه‌داده‌ای شکست بخورد (برای مثال، به دلیل عملکرد نادرست سخت‌افزاری)، تمام تراکنش‌های اجرا شده در بازه خرابی شکست خواهد خورد. به منظور اندازه‌گیری شدت این تأثیر، در دسترس بودن هر عملکرد می‌تواند به عنوان حاصل در دسترس بودن اجزای منفرد درگیر در این عملیات محاسبه شوند. برای مثال، اگر هر پارتیشن پایگاه‌داده ۹۹٫۹٪ در دسترس باشد، یعنی ۴۳ دقیقه در هر ماه می‌تواند در دسترس نباشد، اجرا تثبیت با استفاده از پروتکل PC₂ بر ۲ پارتیشن در دسترس بودن را به ۹۹٫۸٪ کاهش می‌دهد، یعنی ۸۶ دقیقه در ماه خارج از دسترس بودن.

علاوه بر این، PC₂ پروتکلی مسدود کننده است، به این معنی که پایگاه‌داده‌های درگیر در یک تراکنش هنگام روند اجرا تثبیت به صورت موازی قابل استفاده نیستند. در نتیجه، با افزایش تعداد تراکنش‌های همزمان، تأخیر سیستم بیش‌تر خواهد شد. به همین دلیل، در بسیاری از رویکردهای پایگاه‌داده‌های NoSQL، تصمیم گرفته شد که محدودیت‌های سازگاری کاهش داده شوند. این رویکردها با نام BASE شناخته می‌شوند. ایده پیاده‌سازی این مفهوم در سیستم‌ها این است که، به جای شکست کل سیستم، اجازه دهد بخشی از سیستم دچار شکست شود؛ که منجر به دسترسی بیشتر سیستم خواهد شد.

طراحی سیستم‌های BASE، و پایگاه‌داده‌های NoSQL مبتنی بر BASE به صورت خاص، اجرای عملکردهای مشخصی را که رونوشت‌ها (کپی‌هایی از داده‌ها) را در حالت ناسازگار قرار می‌دهند، مجاز می‌دانند. همان‌طور که از نام آن بر می‌آید، سیستم‌های BASE، با معرفی به اصطلاح حالت نرم رونوشت، دسترسی پذیری سیستم را در اولویت قرار می‌دهند، یعنی هر پارتیشن ممکن است دچار شکست شود و به کمک رونوشت‌ها بازسازی شود. از طرفی، این سیستم‌ها مکانیزمی نیز برای همگام‌سازی رونوشت‌ها تعبیه کرده‌اند. این مکانیزم با عنوان Eventual Consistency شناخته می‌شود، تکنیکی که بر اساس برخی معیارهایی سیستم را به حالت سازگار برمی‌گرداند و ناسازگاری‌ها را حل می‌کند. اگرچه Eventual Consistency تضمین نمی‌کند که کاربر در هنگام خواندن دقیقاً مقدار مشابهی از تمام رونوشت‌ها را دریافت کند، اما با

توجه به نوع کاربرد از پایگاه داده‌ها، وجود این درخواست‌های خواندن در بسیاری از کاربردها قابل قبول است. برای مثال پایگاه داده Cassandra در پایگاه داده‌های NoSQL، سیاست‌های زیر را در جهت بروزرسانی رونوشت‌ها پیاده‌سازی می‌کند:

♦ **اصلاح در هنگام خواندن:** ناسازگاری‌ها در طول خواندن داده اصلاح می‌شوند. به این معنی که نوشتن داده ممکن است ناسازگاری‌هایی بر جای بگذارد، که تنها بعد از عملیات خواندن برطرف خواهد شد. در این روند، مولفه دریافت‌کننده، داده را از مجموعه‌ای از رو نوشت‌ها می‌خواند و اگر مقادیر ناسازگار بیابد، موظف است که رونوشت‌هایی که داده‌های قدیمی دارند را بروز رسانی کند. قابل توجه است که تضاد در رونوشت‌ها فقط زمانی رفع خواهد شد که داده‌ها درگیر عملیات خواندن باشند.

♦ **اصلاح در زمان نوشتن:** زمانی که داده‌ها (رونوشت‌ها) بر روی مجموعه‌ای از گره‌ها نوشته می‌شود، ممکن است تعدادی از گره‌ها در دسترس مولفه هماهنگ‌کننده نباشند. با استفاده از این سیاست، به‌روزرسانی‌ها برای زمانی که گره‌ها در دسترس باشند زمان‌بندی می‌شوند تا اجرا شوند.

♦ **اصلاح ناهمگام:** تصحیح کردن نه بخشی از عملیات نوشتن و نه عملیات خواندن است، همگام‌سازی می‌تواند با گذشت زمان از آخرین به‌روزرسانی، مقدار عملیات‌های نوشتن یا رخدادهای دیگری که نشان می‌دهد پایگاه داده در حالت سازگار نیست، آغاز شود.

علاوه بر سازگاری در عملیات‌های خواندن و نوشتن، در سیستم‌های ذخیره‌سازی توزیع شده مفهوم پایایی به‌وجود می‌آید، که توانایی یک سیستم در ماندگاری داده‌ها حتی با وجود شکست‌ها است. به همین دلیل پیش از نمایش پیغام موفقیت عملیات به کاربر، داده‌ها بر روی تعدادی حافظه غیرفرار نوشته می‌شود. در سیستم‌های Eventually Consistent، مکانیزم‌هایی برای درجه‌بندی پایایی و سازگاری سیستم وجود دارد. در ادامه، این مفاهیم را با کمک یک مثال روشن می‌کنیم. فرض کنید N تعداد گره‌هایی است که کلیدی بر روی آن‌ها رونوشت شده است، W تعداد گره‌هایی که برای موفق خواندن عمل نوشتن مورد نیاز باشد، و R تعداد گره‌هایی باشد که عملیات خواندن روی آن‌ها اجرا می‌شود. جدول پایین پیکربندی‌های مختلف برای W و R و همچنین نتایج اعمال این پیکربندی‌ها را نشان می‌دهد. هر مقدار به تعداد رونوشت‌هایی که برای تأیید موفقیت عملیات لازم است اشاره دارد.

سازگاری قوی با اجرای $W+R > N$ اتفاق می‌افتد؛ یعنی مجموعه گره‌های درگیر در عملیات‌های خواندن و نوشتن طوری هم‌پوشانی دارند که همیشه عملیات‌های خواندن آخرین نسخه از داده را به دست می‌آورند. معمولاً RDBMs ها شرط $W=N$ را دارند؛ تمام رونوشت‌ها ماندگار هستند و همچنین دارای شرط $R=1$ هستند چون در هر عملیات خواندن، داده‌های

بروز را بازمی‌گرداند. سازگاری ضعیف زمانی به وجود می‌آید که $W+R \leq N$ برقرار می‌شود. در این شرایط عملیات‌های خواندن ممکن است داده‌های قدیمی دریافت کنند. Eventual Consistency، مورد خاصی از سازگاری ضعیف است که در آن تضمین می‌شود، اگر داده‌ای بر روی سیستم نوشته می‌شود، نهایتاً به تمام رونوشت‌ها فرستاده خواهد شد. این موضوع بستگی به تأخیر شبکه، تعداد رونوشت‌ها و بار سیستم و معیارهای دیگری دارد.

اگر نیاز باشد عملیات‌های نوشتن سریع‌تر صورت بگیرد، می‌توان با قبول پایایی کمتر، روی یک یا مجموعه‌ای کمتر از گره‌ها انجام شوند. اگر $W=0$ باشد، کاربر نوشتن سریع، اما با کمترین پایایی ممکن را تجربه می‌کند، زیرا هیچ تأییدیه‌ای مبنی بر موفقیت آمیز بودن عملیات نوشتن وجود ندارد. اگر $W=1$ ، داده‌ها حداقل باید در یک گره نوشته شوند تا موفقیت آمیز بودن عملیات نوشتن به کاربر بازگردانده شود، و در نتیجه پایایی را به نسبت $W=0$ بهبود می‌بخشد. به صورت مشابه می‌توان عملیات خواندن را بهینه ساخت. انتخاب $R=0$ به عنوان یک گزینه مطرح نیست. برای دستیابی به مقدار بهینه برای خواندن، می‌توان از $R=1$ استفاده کرد. در بعضی مواقع (مثل استفاده از سیاست اصلاح در زمان خواندن)، ممکن است ملزم به خواندن از تمام گره‌ها باشیم، یعنی $R=N$ ، و با وجود کاهش سرعت عملیات، نسخه‌های متفاوت داده را ادغام کنیم. طرحی میانه برای نوشتن یا خواندن، قاعده حدنصاب می‌باشد، به این صورت که عملیات (خواندن یا نوشتن) روی زیرمجموعه‌ای از گره‌ها انجام می‌شود. معمولاً، مقدار استفاده شده در قاعده حدنصاب $N/2 + 1$ است، به طوری که ۲ نوشتن یا خواندن متوالی حداقل در یک گره مشترک باشند.

مقدار	نوشتن (W)	خواندن (R)
۰	انتظار هیچ تأییدیه‌ای از هیچ گره‌ای نداریم (امکان شکست وجود دارد)	بدون کاربرد
۱	یک تأییدیه واحد کافی است (حالت بهینه در نوشتن)	خواندن از طریق یکی از رونوشت‌ها انجام می‌گیرد (حالت بهینه خواندن)
$M < N$ (حدنصاب)	انتظار تأییدیه از تعدادی از رونوشت‌ها را داریم	خواندن از طریق مجموعه‌ای از رونوشت‌ها انجام می‌گیرد (ممکن است نیاز به رفع اختلافات سمت کاربر باشد)
N (تمام گره‌ها)	انتظار تأییدیه از تمام رونوشت‌ها را داریم (دسترسی کاهش می‌یابد، اما پایایی افزایش پیدا می‌کند)	خواندن از طریق تمام رونوشت‌ها صورت می‌گیرد که باعث افزایش تأخیر عملیات خواندن می‌شود

شکل ۳-۳

در پایگاه داده‌های غیررابطه‌ای علاوه بر توجه به ملاحظات سازگاری و دسترس‌پذیری داده‌ها، نوع مدل داده‌ایی که یک پایگاه داده پشتیبانی می‌کند نیز بسیار اهمیت دارد و در تصمیم‌گیری انتخاب یک ابزار مناسب برای ذخیره داده‌ها نقش بسزایی دارد.

ماشین مجازی

۴-۱ معرفی

یک ماشین مجازی، در ابتدا توسط Popek and Goldberg به صورت "یک نسخه کپی شده از روی یک ماشین واقعی، به صورت کارا و ایزوله شده" تعریف شد. استفاده‌های کنونی، ماشین‌های مجازی‌ای را شامل می‌شود که هیچ ارتباط با سخت‌افزار واقعی ندارند.

ماشین‌های مجازی، بر اساس استفاده و درجه ارتباط به ماشین واقعی، به دو دسته اصلی تقسیم می‌شوند.

یک ماشین مجازی سیستمی یک زیرساخت محاسباتی کامل را فراهم می‌کند که از اجرای یک سیستم‌عامل کامل پشتیبانی می‌کند.

در مقابل، یک ماشین مجازی فرایند، برای اجرای یک برنامه واحد طراحی شده، که این به این معناست که صرفاً از یک فرایند خاص پشتیبانی می‌کند.

یک ویژگی مهم یک ماشین مجازی، این است که نرم‌افزاری که درون آن در حال اجراست، با منابع و سطوح انتزاعی که توسط ماشین مجازی اعمال می‌شود، محدود شده‌است – یعنی نمی‌تواند از دنیای مجازی خود خارج شود.

نرم افزار vmware workstation ابزاری مفید است که این امکان را برای شما فراهم می‌آورد تا چندین سیستم عامل مختلف را روی ماشین خود نصب کنید و این توانایی را داشته باشید که بین سیستم عامل های مختلف بر حسب نیاز سوئیچ کرده و از مزایای OS های مختلف بهره مند شوید. یکی از مواردی که ما را ناگزیر به استفاده از این نرم افزار می‌کند نیاز به استفاده از سیستم عامل لینوکس است، خصوصاً برای کسانی که کاربر ویندوز هستند.



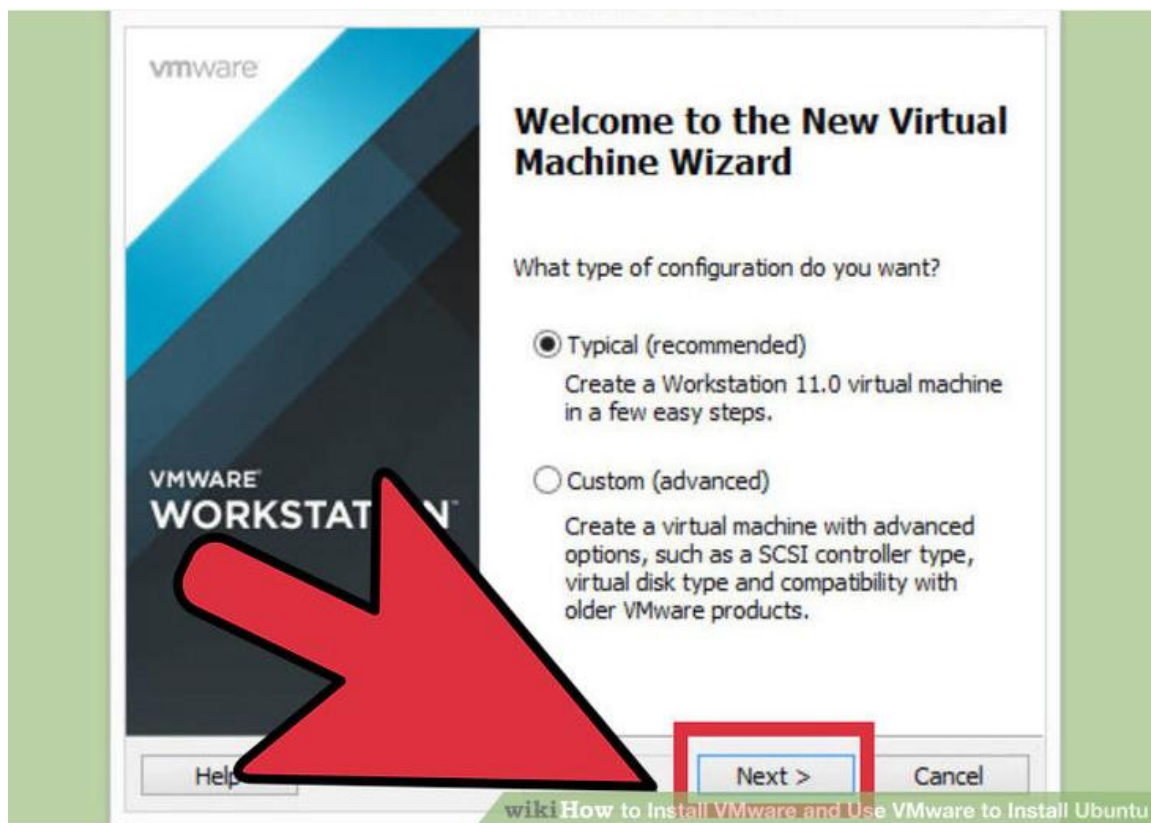
شکل ۴-۱

۴-۲ آموزش نصب

آخرین نسخه نرم افزار vmware workstation را دریافت کرده و نصب می کنیم.

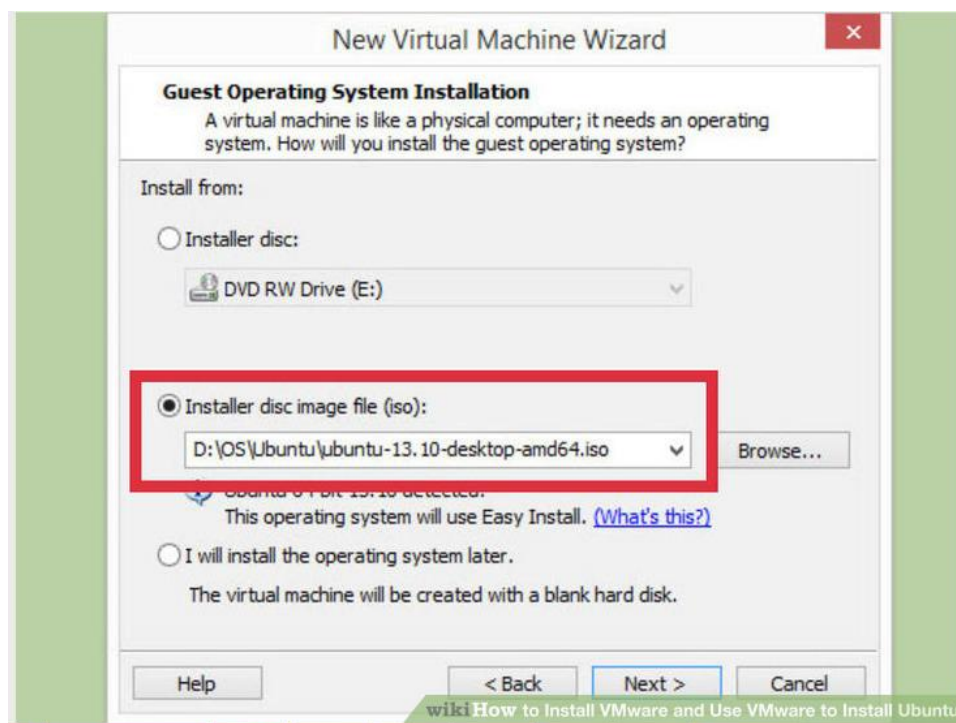
پس از نصب نرم افزار، نسخه مورد نظر از سیستم عامل ubuntu را دریافت کرده و اقدام به نصب آن بر روی ماشین مجازی می کنیم:

۱- گزینه Typical را انتخاب می کنیم.



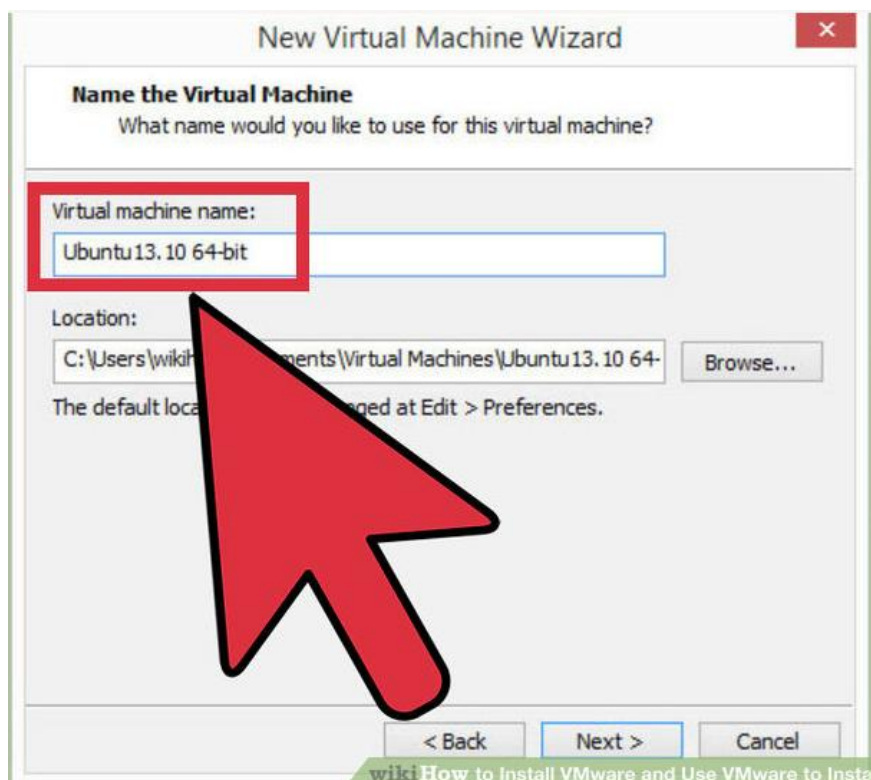
شکل ۴-۲

۲- آدرس فایل ایزوی Ubuntu را به vmware می دهیم.



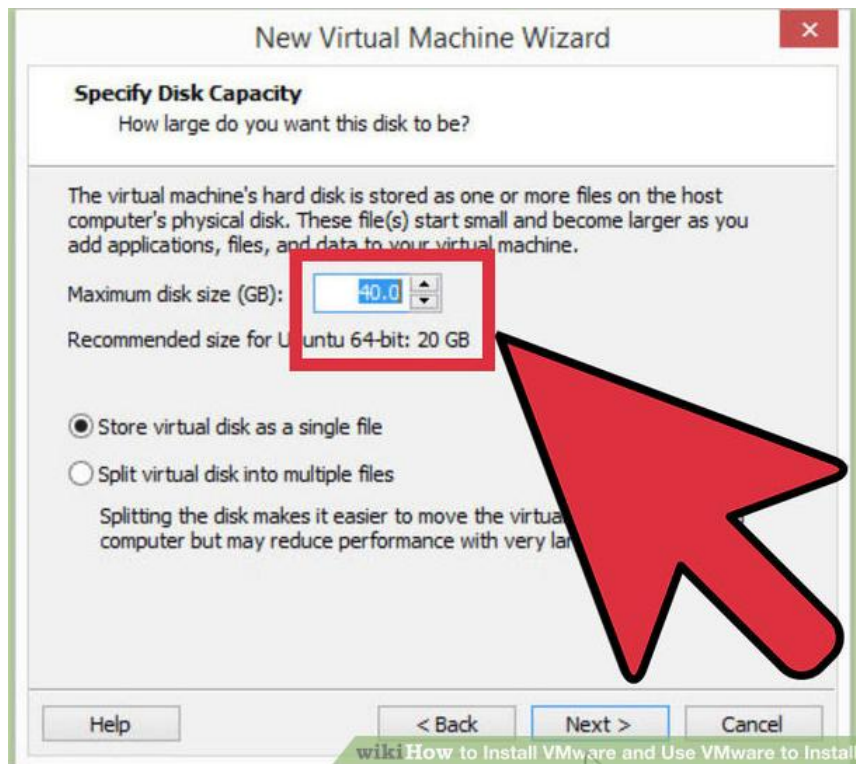
شکل ۴-۳

۳- نام ماشین مجازی و محل ذخیره سازی آن را بر روی سیستم خود انتخاب می کنیم.



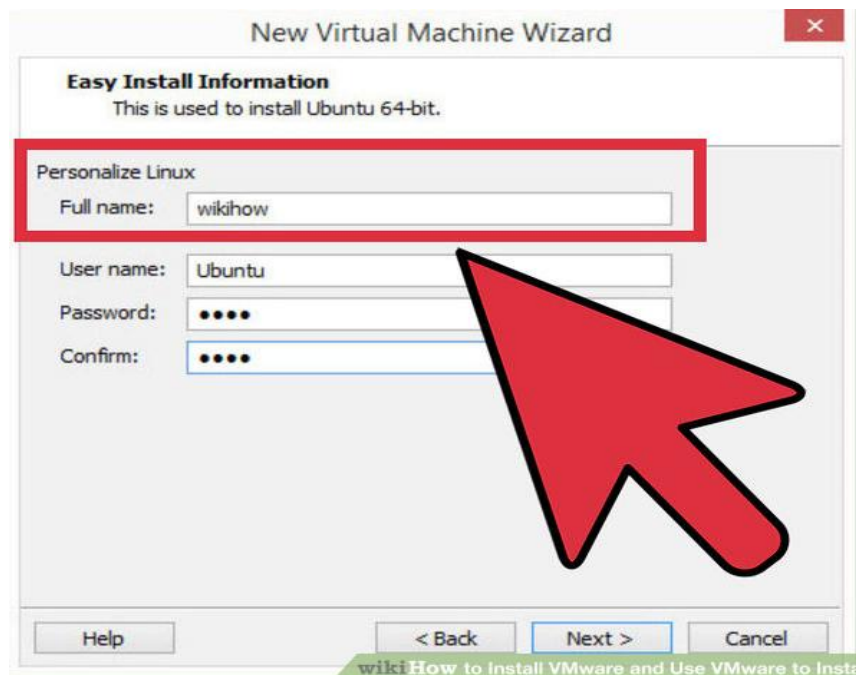
شکل ۴-۴

۴- میزان فضای لازم را بر حسب نیاز، از هارد خود به ماشین اختصاص می دهیم.



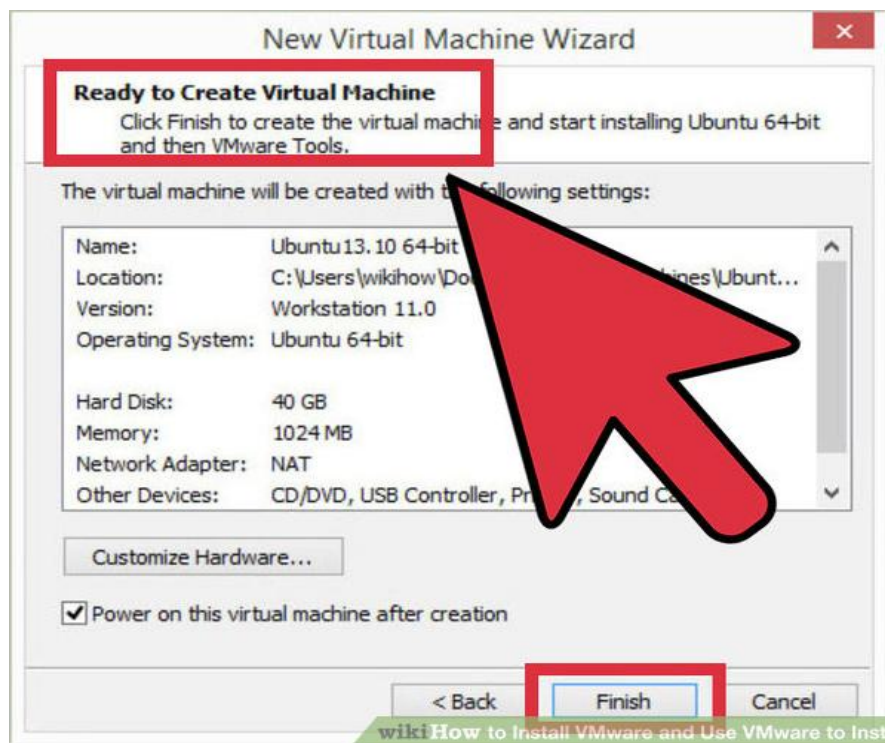
شکل ۴-۵

۵- نام کامپیوتر، نام کاربر و رمز عبور خود را تنظیم می کنیم.



شکل ۴-۶

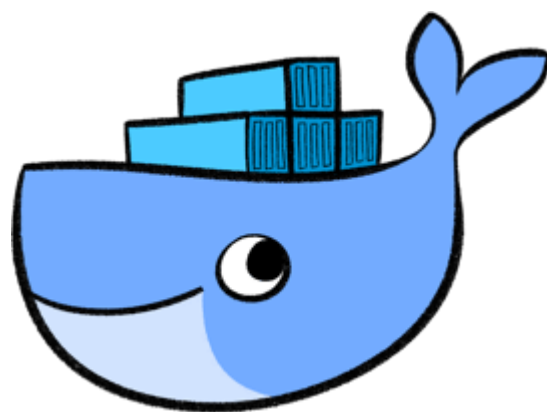
۶- اگر نیاز به تنظیمات خاص سخت افزاری برای ماشین مجازی خود هستید می توانید از طریق Customized Hardware این کار را انجام دهید. در غیر اینصورت با فشردن دکمه Finish کار ساخت ماشین مجازی را به پایان ببرید.



شکل ۴-۷

پس از طی کردن مراحل فوق، اقدام به اجرای ماشین مجازی می کنیم و در اولین اجرا سیستم عامل اوبونتو را نصب می کنیم.

داکر-DOCKER



شکل ۵-۱

Docker یک سکوی متن باز برای ساخت، طراحی و اجرای اپلیکیشن‌های توزیع شده است. توسط داکر می‌توانید اپلیکیشن‌های خود را سریع تر و راحت تر منتشر کنید. Docker به صورتی عمل می‌کند که عملیات بسته بندی، حمل و توسعه هر برنامه کاربردی که به صورت سبک و قابل حمل ایجاد شده است را بطور خودکار انجام دهد. از دیگر ویژگی‌های این پلتفرم، امکان گسترش سرویس‌های قابل توسعه به شیوه ای امن و قابل اعتماد در طیف گسترده ای از پلتفرم‌ها است. از دیگر قابلیت‌های داکر می‌توان به انتقال اپلیکیشن‌ها و انعطاف پذیری زیر ساخت، به روز رسانی پویا و ایجاد تغییرات در لحظه را نام برد.

داکر هیچ سیستم عامل جدیدی ایجاد نمی‌کند. بلکه این امکان را به بسته نرم افزاری ایجاد شده می‌دهد که از Kernel اصلی سیستم عاملی که بر روی آن نصب شده است استفاده نماید و در زمان انتقال نیز فقط Package نرم افزاری منتقل می‌شود نه ماشین مجازی، در واقع Docker Engine یا موتور اصلی Docker جایگزین نرم افزار Hypervisor می‌شود و اینکار باعث می‌گردد که کارایی سیستم ما به شدت افزایش یابد. زیرا یک لایه واسطه به نام Hypervisor حذف شده و نرم افزار بصورت مستقیم با هسته اصلی سیستم عامل کار می‌کند. با این تفاوت که کاملاً ایزوله شده است.

یکی از مهمترین فاکتورهایی که Docker دارد Open Source بودن آن است. متن باز بودن بدین معنی می‌باشد که هر کسی می‌تواند Docker را تهیه و سورس آن را تغییر بدهد و یک محصول جدید معرفی کند و یا اینکه قابلیت‌های جدیدی به آن اضافه کند که تا به حال بر روی آن وجود نداشته است. شما می‌توانید Docker Container های مختلفی بر روی یک سیستم پیاده سازی کرده، بطوریکه تمامی آنها در یک فضای ایزوله شده قرار داشته و از سیستم میزبان هم مجزا باشند. با استفاده از این پلتفرم، می‌توانید کل چرخه توسعه، تست، توزیع و مدیریت را با استفاده از رابط کاربری مستحکم طراحی کنید.

تفاوت Docker و Virtual Machine ها



شکل ۵-۲

۵-۳ داکر برای چه کسانی مناسب است؟

داکر ابزاری است که هم برای برنامه‌نویس‌ها و هم به مدیران شبکه مناسب است و به همین خاطر هم برخی اوقات به نام DevOps از آن یاد می‌شود که ترکیبی از دو اسم Developer و Operations است. برای برنامه‌نویس‌ها Docker به این معنا است که فقط روی کدنویسی خودتان تمرکز کنید و دغدغه اینکه کد برنامه شما قرار است بر روی چه سیستم‌عاملی با چه نیازمندی‌هایی نصب شود را نداشته باشید. این کار را Docker برای شما انجام می‌دهد. از طرفی هزاران برنامه و نرم‌افزار متنوع وجود دارند که برای کار کردن در محیط Docker طراحی شده‌اند و شما به عنوان یک متخصص IT می‌توانید به راحتی از آنها در مجموعه خودتان در قالب یک Docker Container استفاده کنید. از طرفی در محیط‌های عملیاتی Docker این امکان را به همه می‌دهد که چندین برنامه را همزمان بر روی یک سیستم فیزیکی نصب و اجرا کنند و اینها هیچکدام با یکدیگر کوچکترین ارتباطی نداشته باشند و بصورت کاملاً ایزوله در مجموعه فعالیت نمایند.

۴-۵ مکانیزم کاری DOCKER چگونه است؟

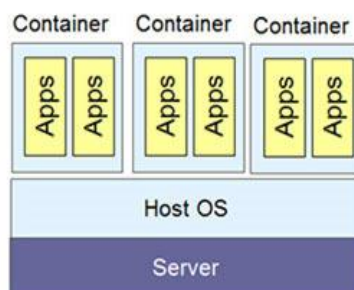
داکر یک لایه واسط بین سیستم عامل اصلی شما و بسته نرم افزاری تان ایجاد کرده و در واقع با استفاده از این لایه واسط نرم افزارها را از همدیگر ایزوله می کند، هیچکدام از نرم افزارها از وجود نرم افزار دیگر بر روی سیستم خبری ندارند. این مکانیزم یک چیز عجیب و غریب برای لینوکس نیست، در سیستم عامل لینوکس قابلیت هایی برای ایزوله سازی منابع وجود داشته و دارند که هم هسته سیستم عامل و هم گروه ها و منابع سخت افزاری و نرم افزاری سیستم عامل را بصورت ایزوله شده در اختیار نرم افزارها قرار می دهند که Docker نیز از آنها استفاده می کند. برای مثال قابلیت های cgroups و kernel namespaces از جمله مواردی هستند که Docker از آنها برای کار خودش استفاده می کند. قابلیتی مثل kernel namespace باعث می شود که application ها هیچ دیدی از محیطی که در آن اجرا می شوند نداشته باشند که این موارد شامل: process tree ها، شبکه، ID های کاربران و حتی فایل سیستم های mount شده نیز می شود. از طرفی قابلیتی مثل cgroups محدودیت های دسترسی به منابع: CPU، RAM، I/O و شبکه را ایجاد می کند. Docker در محیط های اشتراکی (Shared Enviroment) امنیت را نیز برای نرم افزارهای ما به ارمغان می آورد. اما به عنوان یک مکانیزم امنیتی شناخته نمی شود. شما به عنوان یک برنامه نویس یا شبکه کار بایستی سیستم عامل Docker را بصورت جداگانه امن کنید.

۵-۵ داکر هاب

یک سرویس اشتراک گذاری تهیه شده توسط شرکت Docker است که شامل مخزنی از image های آماده برای Docker است.

این مخزن حاوی ده ها هزار برنامه و سیستم عامل است که می توان به آن image هایی را هم اضافه کرد.

۶-۵ کانتینر داکر



شکل ۳-۵

در واقع می‌توان گفت Container ظرفی است که Image ها را در آن اجرا می‌کنند. Container ها از روی Image ها ایجاد می‌شوند و به وظایف خود عمل می‌کنند. مثلاً فرض کنید از یک Centos چند Container می‌سازیم و در هر کدام تغییرات متفاوتی اعمال می‌کنیم.

۷-۵ نصب داکر بر روی اوبونتو

برای انجام دستورات زیر نیاز دارید که دسترسی root داشته باشید. از آنجا که بسته (package) موجود در مخزن اوبونتو ۱۶.۰۴ برای نصب داکر ممکن است آخرین نسخه نباشد، پیشنهاد می‌شود آخرین نسخه را از مخزن رسمی داکر دریافت کنید.

برای این کار ابتدا اطلاعات تمام بسته‌ها را بروز رسانی کنید:

```
sudo apt-get update
```

برای نصب داکر، کلید GPG مخصوص مخزن رسمی داکر را به سیستم خود اضافه کنید:

```
sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys
```

```
58118E89F3A912897C070ADB7F6221572C52609D
```

سپس مخزن داکر را به منابع APT اضافه کنید تا بتوانید با کمک دستور apt-get بسته‌های این مخزن را نصب کنید:

```
sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial main'
```

در قدم بعدی، دوباره اطلاعات بسته‌ها را که این بار شامل مخزن جدید داکر می‌شود بروز رسانی کنید:

```
sudo apt-get update
```

قبل از نصب مطمئن شوید که موتور داکر را از مخزن پروژه داکر دانلود می‌کنید، نه از مخزن پیش فرض اوبونتو. برای این کار دستور زیر را وارد کنید:

```
apt-cache policy docker-engine
```

و می‌بایست خروجی زیر را مشاهده کنید:

```
docker-engine: Installed: (none) Candidate: ۱,۱۱,۱-۰~xenial Version table: ۱,۱۱,۱-
```

```
۰~xenial ۵۰۰ ۵۰۰ https://apt.dockerproject.org/repo ubuntu-xenial/main amd64
```

ubuntu- <https://apt.dockerproject.org/repo> ۵۰۰ xenial ۵۰۰ Packages ۱,۱۱,۰۰۰~xenial ۵۰۰ Packages

xenial/main amd64 Packages

این پیام نشان میدهد که گزینه انتخابی برای نصب، مخزن پروژه داکر است. البته نسخه نشان داده شده برای شما ممکن است متفاوت باشد ولی آدرس مخزن باید حتما در سایت apt.dockerproject.org باشد.

در نهایت، موتور داکر را نصب کنید:

```
sudo apt-get install -y docker-engine
```

زمانی که عملیات نصب به پایان رسید، موتور داکر به صورت خودکار اجرا شده و پردازش داکر برای اجرا در هنگام راه اندازی سیستم (boot) فعال شده است. با دستور زیر می توانید وضعیت اجرایی آن را چک کنید:

```
sudo systemctl status docker
```

نتایج باید مشابه زیر باشد که وضعیت سرویس را در حال اجرا و فعال نشان میدهد (Active & running):

```
docker.service - Docker Application Container Engine Loaded: loaded
(/lib/systemd/system/docker.service; enabled; vendor preset: enabled) Active: active
(running) since Sun ۲۰۱۷-۱۰-۰۱ ۰۶:۵۳:۵۲ CDT; ۱ weeks ۳ days ago Docs:
https://docs.docker.com Main PID: ۷۴۹ (docker)
```

در این مرحله نصب داکر تمام شده است.

اجرای دستورات داکر بدون نیاز به `sudo`

برای اجرای دستورات داکر می بایست آنها را با کمک دستور `sudo` اجرا کنید و در واقع نیاز به سطح دسترسی `root` دارید. اگر تمایل دارید بتوانید این دستورات را بدون `sudo` اجرا کنید میتوانید نام کاربری خود را به گروه داکر که در حین نصب ساخته شده است، اضافه کنید.

برای این منظور دستور زیر را پس از جایگزین کردن نام کاربری مورد نظرتان اجرا کنید:

```
sudo usermod -aG docker USERNAME
```

حال میتوانید دستورات داکر را بدون نیاز به `sudo` اجرا کنید، ولی دقت کنید که این کار در عمل به هر کاربری که عضو گروه `docker` شده باشد سطح دسترسی `root` میدهد.

۸-۵ استفاده از داکر برای پروژه

پس از نصب آن بر روی سیستم، یک image از اوبونتو را از داکر هاب دریافت می کنیم و بر روی آن، هدوپ و سایر موارد را نصب می کنیم. برای این منظور به صورت زیر عمل می کنیم.

۱- با توجه به فیلتر بودن سایت داکر هاب، از سایت های ایرانی موجود استفاده کرده و ایمج مورد نظر را دریافت می کنیم:

<http://elastico.io/>

`docker pull hub.elastico.io/library/image-name`

نام image مورد نظر را در انتهای دستور فوق قرار داده (به جای image-name) تا دریافت صورت پذیرد.

```
saeed@saeed-X550CC:~$ docker pull hub.elastico.io/library/ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
50aff78429b1: Downloading [====>] 2.632 MB/42.74 MB
f6d82e297bce: Download complete
275abb2c8a6f: Download complete
9f15a39356d6: Download complete
fc0342a94c89: Download complete
```

شکل ۴-۵

۲- پس از دریافت کامل image، می توان با دستور زیر لیست Image های موجود در سیستم خود را ببینید

`docker images`

```
saeed@saeed-X550CC:~$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
ubuntu-hadoop       latest      404a3f5d032a     2 weeks ago     1.02 GB
ubuntu              latest      4187f8fa914a     2 weeks ago     541 MB
hub.elastico.io/library/ubuntu latest      747cb2d60bbe     2 months ago    122 MB
hadoop-img          latest      d6773739675e     2 months ago    5.58 GB
docker.loc:5000/server 1.0-g       09509117d4e3     6 months ago    1.27 GB
```

شکل ۵-۵

۳- برای ایجاد یک کانتینر (container) داکر از دستور زیر به همراه آرگومان های ورودی آن استفاده می کنیم

`docker run -it -v somewhere/./somewhere --net=host -p port-number:port-number --name test docker-image`

در آن محل قرارگیری کانتینر، شماره پورت متناظر با آن، اسم آن و image مورد استفاده باید مشخص شود. برای اینکه کانتینر ایجاد شده به نت سیستم میزبان دسترسی داشته باشد از آرگومان زیر استفاده می کنیم

`--net=host`

۴- برای مشاهده لیست تمام کانتینر های موجود در سیستم از دستور زیر استفاده می کنیم:

`docker ps -a`

```
saeed@saeed-X550CC:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b80aa7e94668	ubuntu	"/bin/bash"	2 weeks ago	Exited (0) 2 weeks ago		hive
64710cac73bc	ubuntu	"/bin/bash"	2 weeks ago	Exited (0) 2 weeks ago		hbase
ec1c43ba0c4c	ubuntu	"/bin/bash"	2 weeks ago	Exited (0) 2 weeks ago		hadoop
0c475c824cc4	hub.elastico.io/library/ubuntu	"/bin/bash"	3 weeks ago	Exited (0) 3 weeks ago		spark
48b40c444b45	docker.loc:5000/server:1.0-g	"/bin/bash"	3 months ago	Exited (0) 2 weeks ago		coder

```
saeed@saeed-X550CC:~$
```

شکل ۵-۶

برای مشاهده کانتینرهای فعال از دستور زیر استفاده می کنیم:

`docker ps`

```
saeed@saeed-X550CC:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ec1c43ba0c4c	ubuntu	"/bin/bash"	2 weeks ago	Up 3 seconds		hadoop

```
saeed@saeed-X550CC:~$
```

شکل ۵-۷

۵- برای فعال کردن یک کانتینر از دستور زیر استفاده می کنیم

`docker start hadoop`

۶- برای وارد شدن به محیط کانتینر از دستور زیر استفاده می کنیم

`docker attach hadoop`

۷- برای غیر فعال کردن کانتینر از دستور `exit` در داخل کانتینر و یا دستور `stop` در خارج از آن می شود استفاده کرد.

آپاچی ہڈوپ

APACHE
HADOOP

هدوپ یک ابزار متن‌باز از بنیاد نرم‌افزار آپاچی است. پروژه متن‌باز به این معنا است که بدون محدودیت در دسترس است و حتی کد منبع آن نیز طبق نیاز می‌تواند تغییر داده شود. اگر یک قابلیت خاص نمی‌تواند نیاز شما را برآورده کند، می‌توانید آن را بر اساس نیاز خود تغییر دهید. بسیاری از کدهای هدوپ توسط یاهو، آی‌بی‌ام، فیس‌بوک و کلودرا نوشته شده است.

این ابزار یک چارچوب کارآمد برای اجرای برنامه بروی کلاسترها را فراهم می‌آورد. کلاستر به معنای گروهی از سیستم‌ها است که از طریق یک شبکه ارتباطی به هم متصل شده‌اند. هنگامی که مولفه‌های هدوپ به طور هم‌زمان روی ماشین‌های متعدد مستقر می‌شوند، پردازش و یا رایانش موازی داده‌ها را برای ما به ارمغان می‌آورد.

پروژه هدوپ با الهام از دو ابزاری که گوگل از آن برای موتور جستجوی خود استفاده می‌کند، ایجاد شده است. گوگل در سال ۲۰۰۳ مقاله‌ای در مورد سیستم فایل توزیع شده خود بنام GFS و مدل برنامه نویسی نگاشت/کاهش منتشر کرد. توسعه هدوپ در ابتدا در پروژه Nutch، که آن زمان Doug Cutting و تیم او مشغول کار روی آن بودند، شکل گرفت؛ اما با توجه به محبوبیت بسیار زیادی که داشت، خیلی زود به یک پروژه‌ی سطح بالای آپاچی تبدیل شد.

هدوپ یک چارچوب منبع‌باز است که به زبان جاوا نوشته شده است. اما این بدان معنا نیست که برنامه‌نویسی برای آن تنها به زبان جاوا امکانپذیر باشد. می‌توان به زبان‌های C، ++C، پرل، پایتون و روبی برنامه‌های مربوط به این سکو را توسعه داد. اما اگر بدنبال استفاده از تمام خصوصیات توسعه داده شده در هدوپ هستید و یا نیاز است که در سطوح پایین نیز کنترلی بروی کد خود داشته باشید، توصیه می‌شود از زبان جاوا برای توسعه برنامه خود استفاده کنید.

یکی از ویژگی‌های دیگر پروژه هدوپ این است که این ابزار حجم انبوه داده‌ها را در شبکه‌ای از سیستم‌های سخت‌افزاری معمولی را به طور کارآمد پردازش می‌کند. هدوپ به منظور پردازش حجم انبوهی از داده‌ها ساخته شده است. منظور از سخت‌افزارهای معمولی آن دسته از سخت‌افزارهایی هستند که معمولاً در مراکز داده بکار می‌روند و برای استقرار هدوپ نیاز به یک سخت‌افزار خاص منظوره نیستیم. در نتیجه هدوپ نیز بسیار به صرفه است.

هدوپ می‌تواند روی یک تک دستگاه (حالت شبه توزیع شده) نصب شود، اما توانایی واقعی هدوپ در کلاستر یا شبکه‌ای از دستگاه‌ها همراه است و می‌تواند با قابلیت اطمینان بالا بروی کلاستری با هزاران ماشین در حال کار اجرا شود.

هدوپ دارای سه بخش کلیدی است: سیستم فایل توزیع شده هدوپ (HDFS)، چهارچوب برنامه نویسی نگاشت/کاهش و ابزار مدیریت منابع Yarn.

۲-۶ چرا هدوپ؟

هدوپ نه تنها یک سیستم توزیع شده ذخیره سازی داده است بلکه علاوه بر آن یک سکوی پردازش داده نیز در اختیار ما میگذارد. همچنین هدوپ مقیاس پذیر است (سیستم‌های بیشتر می‌توانند بدون توقف فرایند اضافه شوند)، تحمل‌پذیر در مقابل خطا است (حتی اگر یکی از سیستم‌ها دچار مشکل شود، پردازش دیتا می‌تواند توسط سیستم دیگر انجام شود) و متن باز است (در صورت نیاز کد منبع می‌تواند تغییر داده شود).

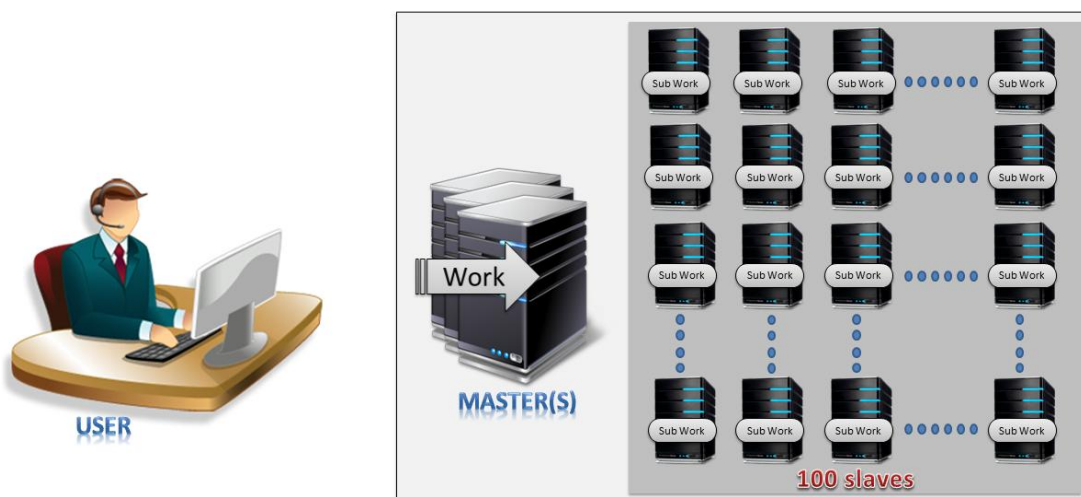
برخی از خصوصیتی که باعث شده تا هدوپ به عنوان یک ابزار محبوب برای محیط های کلان داده مطرح شود، عبارتند از :

- ♦ انعطاف پذیری در ذخیره سازی داده ها و امکان ذخیره سازی داده ها به هر نوع فرمت اعم از ساخت یافته، نیمه ساخت یافته و بدون ساختار. در واقع در هنگام ذخیره سازی داده محدود به نوع خاصی نیستیم.
- ♦ انجام پردازش های پیچیده توسط موتور پردازشی نگاشت/کاهش و توزیع آنها بین ماشین های کلاستر. در سکوی هدوپ کدهای برنامه به سمت داده ها ارسال می شوند در مقابل آنکه داده ها به سمت برنامه در حال اجرا ارسال شوند. این موضوع باعث استفاده بهینه از پهنای باند کلاستر شده و از ایجاد بار اضافی ناشی از انتقال داده ها در بین گره های کلاستر جلوگیری می کند.
- ♦ از نظر صرفه اقتصادی، همان طور که گفته شده، هدوپ می تواند روی یک دستگاه معمولی نصب شود. جدا از این موضوع، برخورداری از ویژگی متن باز بودن، آن را در برابر وابستگی به فروشنده حفظ می کند.

۳-۶ معماری هدوپ

هدوپ به سبک Master/Slave کار می کند. در کلاستر هدوپ یک گره Master وجود دارد و تعداد زیادی گره Slave که این تعداد می تواند تا هزاران گره نیز ادامه پیدا کند. گره Master، گره های Slave را مدیریت، حفظ و کنترل می کند در حالی که گره های Slave، مولفه های واقعی انجام کار هستند.

گره Master، فقط فراداده ها (داده های درباره داده) را ذخیره می کند در حالی که Slave ها گره هایی هستند که داده ها را ذخیره می کنند. ذخیره سازی داده بین کلاستر توزیع می شود. کارخواه یا کلاینت برای انجام هر کار به گره Master متصل می شود.



شکل ۶-۱

۶-۴ سیستم فایل توزیع شده هدوپ (HDFS)

HDFS یک سیستم فایل توزیع شده است که امکان ذخیره‌سازی در هدوپ را به یک شیوه توزیع شده فراهم می‌آورد. در گره Master یک سرویس به نام NameNode اجرا می‌شود. همچنین بروی همه گره‌های Slave سرویسی به نام DataNode برای HDFS اجرا می‌شوند. NameNode فراداده‌ها را ذخیره و هنگامی که گره‌های داده، دیتاها را ذخیره می‌کنند و وظیفه واقعی را انجام می‌دهند، آن‌ها را مدیریت می‌کند.

HDFS یک سیستم فایل بسیار تحمل‌پذیر، توزیع شده، قابل اعتماد و مقیاس‌پذیر برای ذخیره داده‌ها است. در سیستم فایل توزیع شده هدوپ اطلاعات بیشتری در مورد خصوصیات HDFS وجود دارد.

HDFS به منظور پردازش و کنترل حجم انبوه داده‌ها ساخته شده است. اندازه فایل‌های مورد انتظار می‌تواند بین محدوده گیگابایت تا ترابایت باشد. در سیستم فایل HDFS، مشابه اکثر سیستم‌فایل‌ها، از مفهوم بلاک در هنگام ذخیره‌سازی داده‌ها استفاده می‌شود و یک فایل به چند بلاک (به طور پیش‌فرض ۱۲۸ مگابایت) تقسیم و بین گره‌های متعدد ذخیره می‌شود. همچنین به منظور فراهم قابلیت دسترس‌پذیری بالا از مکانیسم تکرار داده‌ها استفاده می‌شود. به همین دلیل بلاک‌های حاصل از یک فایل داده نیز بر اساس هر عامل تکرار (عامل تکرار پیش‌فرض ۳) تکرار و در گره‌های مختلف ذخیره می‌گردند. این عمل باعث می‌شود با ایجاد مشکل در یک گره نیز داده‌ها از دست نروند. بنابراین برای مثال فایلی با حجم ۶۴۰ مگابایت، به ۵ بلاک ۱۲۸ مگابایتی شکسته خواهد شد و هریک از بلاک‌ها در سه گره ذخیره می‌شوند (اگر از مقدار پیش‌فرض استفاده شده باشد).

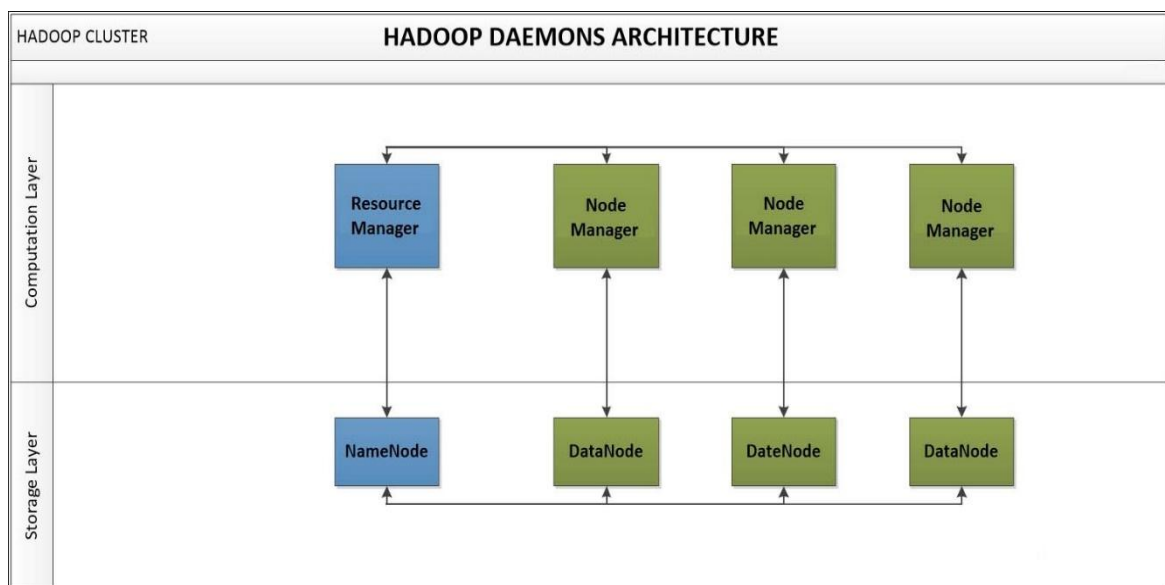
۵-۶ موتور پردازش نگاشت/کاهش

نگاشت/کاهش یک مدل برنامه‌نویسی شناخته شده برای پردازش حجم زیاد داده از طریق تقسیم کار به مجموعه‌ای از کارهای مستقل و اجرای آن بروی گره‌های کلاستر، طراحی شده است. در این مدل کاربر با پیاده‌سازی توابع نگاشت و کاهش الگوریتم مورد نظر خود را پیاده‌سازی می‌کند. علاوه بر این، نگاشت/کاهش موتور پردازشی هادوپ نیز است و از آنجا که انتقال حجم انبوه داده‌ها پر هزینه است، پردازش را به سمت داده‌ها ارسال می‌کند. این موتور پردازشی امکان مقیاس پذیری گسترده میان صدها یا هزاران گره در یک کلاستر هادوپ را ارائه می‌کند و با توجه به اینکه داده‌ها به شیوه‌ای توزیعی در HDFS ذخیره می‌شوند، این عمل شرایطی فراهم می‌آورد تا نگاشت/کاهش پردازش موازی را در بین گره‌های یک کلاستر محقق کند.

۶-۶ سرویس مدیریت منابع Yarn

ابزار مدیریت منابع آپاچی Yarn، لایه مدیریت منابع هادوپ است. در کلاستر چند گره‌ای، کنترل، تخصیص و انتشار منابع (پردازنده، حافظه، دیسک) بسیار پیچیده می‌شود. Yarn به صورت کاملاً کارآمد منابع را مدیریت کرده و با توجه به درخواست برنامه‌ها منابع مورد نیاز را با آنها تخصیص می‌دهد.

در گره Master سرویس ResourceManager و در گره‌های Slave نیز سرویس NodeManager اجرا می‌شود.



شکل ۶-۲

۶-۷ سرویس های هدوپ

۴ سرویس زیر، سرویس هایی هستند که برای استقرار هدوپ بروی یک کلاستر به اجرا در می آیند. سرویس ها پردازش هایی هستند که در پس زمینه اجرا می گردند و به درخواست های رسیده به سمت آنها پاسخ می دهند.

نام سرویس	گره	توضیح
NameNode	Master	استقرار HDFS
DataNode	Slaves	
ResourceManager	Master	استقرار Yarn
NodeManager	Slaves	

شکل ۶-۳

به منظور راه اندازی هدوپ، چهار سرویس اشاره شده در بالا حتما باید اجرا شوند. اما علاوه بر این سرویس ها، سرویس هایی نظیر NameNode Secondary، Standby NameNode، Job History Server و غیره می توانند وجود داشته باشند که هریک وظایفی دارند که بنا بر نیاز می توانند مورد استفاده قرار بگیرند.

۶-۸ مراحل پردازش داده ها توسط هدوپ

اگر بخواهیم مراحل پردازش داده ها توسط هدوپ را بطور خلاصه بررسی کنیم، می توان به مراحل زیر اشاره کرد:

- ♦ **مرحله ۱** در ابتدا داده های مورد نیاز برنامه بروی سیستم فایل HDFS قرار می گیرند. داده های ورودی به بلوک هایی با حجم پیش فرض شکسته شده و در گره های مختلف و با تعداد تکرار مشخص شده ذخیره می شوند.
- ♦ **مرحله ۲** برنامه توسعه داده شده کاربر از طریق ابزار مدیریت منابع Yarn بروی کلاستر توزیع شده و اجرا می شود.
- ♦ **مرحله ۳** زمانی که کار پردازش پردازش داده ها به اتمام رسید، خروجی برنامه بروی HDFS نوشته می شود.

طریقه نصب هدوپ در اوبونتو

۷-۱ مراحل نصب

برای نصب هدوپ، ترمینال اوبونتو را باز کرده و دستورات زیر را به ترتیب اجرا می کنیم:

۱-نصب جاوا

```
sudo apt-get update
```

```
sudo apt-get install default-jdk
```

۲-افزودن کاربر اختصاصی هدوپ

```
sudo addgroup hadoop
```

```
sudo adduser --ingroup hadoop hduser
```

۳-نصب ssh

```
sudo apt-get install ssh
```

```
which ssh
```

```
which sshd
```

۴-ساخت و نصب گواهی های ssh

```
su hduser
```

```
ssh-keygen -t rsa -P ""
```

```
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

۵-نصب هدوپ

```
wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-۲,۷,۴/hadoop-۲,۷,۴.tar.gz
```

```
tar xvzf hadoop-۲,۷,۴.tar.gz
```

```
sudo mv * /usr/local/hadoop
```

```
su saeed
```

```
sudo adduser hduser sudo
```

```
sudo su hduser
```

```
sudo mv * /usr/local/hadoop
```

```
sudo chown -R hduser:hadoop /usr/local/hadoop
```

۶-تنظیم کردن فایل های پیکربندی

فایل های زیر نیازمند ویرایش هستند:

```
~/bashrc
```

```
/usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

```
/usr/local/hadoop/etc/hadoop/core-site.xml
```

```
/usr/local/hadoop/etc/hadoop/mapred-site.xml.template
```

```
/usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

دستور زیر را وارد می کنیم:

```
update-alternatives --config java
```

حال فایل هایی که اشاره شد را به صورت زیر تغییر می دهیم:

۱- فایل bashrc را با دستور vi ~/bashrc باز کرده و کدهای زیر را به انتهای آن اضافه می کنیم:

```
#HADOOP VARIABLES START
```

```
export JAVA_HOME=/usr/lib/jvm/java-۸-openjdk-amd۶۴
```

```
export HADOOP_INSTALL=/usr/local/hadoop
```

```
export PATH=$PATH:$HADOOP_INSTALL/bin
```

```
export PATH=$PATH:$HADOOP_INSTALL/sbin
```

```
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
```

```
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
```

```
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
```

```
export YARN_HOME=$HADOOP_INSTALL
```

```
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END
```

سپس فایل را بسته و برای اعمال تنظیمات دستور زیر را اجرا می کنیم:

```
source ~/.bashrc
```

۲- فایل `hadoop-env.sh` را با دستور `vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh` باز کرده و تغییر زیر را اعمال می کنیم:

```
export JAVA_HOME=/usr/lib/jvm/java-۸-openjdk-amd۶۴
```

۳- برای فایل `core-site.xml` ابتدا دستورات زیر را اجرا می کنیم:

```
sudo mkdir -p /app/hadoop/tmp
```

```
sudo chown hduser:hadoop /app/hadoop/tmp
```

و سپس تغییرات آن را به صورت زیر با دستور `vi /usr/local/hadoop/etc/hadoop/core-site.xml` اعمال می کنیم:

```
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>A base for other temporary directories.</description>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:۵۴۳۱۰</value>
<description>The name of the default file system. A URI whose
```

scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.</description>

</property>

</configuration>

۴- برای فایل بعدی ابتدا دستور زیر را اجرا می کنیم:

```
cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template  
/usr/local/hadoop/etc/hadoop/mapred-site.xml
```

سپس تغییرات زیر را لحاظ می کنیم:

<configuration>

<property>

<name>mapred.job.tracker</name>

<value>localhost:۵۴۳۱۱</value>

<description>The host and port that the MapReduce job tracker runs at. If "local", then jobs are run in-process as a single map and reduce task.

</description>

</property>

</configuration>

۵- برای فایل hdfs-site.xml به صورت زیر عمل می کنیم:

```
sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode  
sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
```

```
sudo chown -R hduser:hadoop /usr/local/hadoop_store
```

```
vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

و کدهای زیر را در آن جای گذاری می کنیم:

```
<configuration>
```

```
<property>
```

```
<name>dfs.replication</name>
```

```
<value>1</value>
```

```
<description>Default block replication.
```

The actual number of replications can be specified when the file is created.

The default is used if replication is not specified in create time.

```
</description>
```

```
</property>
```

```
<property>
```

```
<name>dfs.namenode.name.dir</name>
```

```
<value>file:/usr/local/hadoop_store/hdfs/namenode</value>
```

```
</property>
```

```
<property>
```

```
<name>dfs.datanode.data.dir</name>
```

```
<value>file:/usr/local/hadoop_store/hdfs/datanode</value>
```

```
</property>
```

```
</configuration>
```

پس از انجام تغییرات، فایل سیستم هادوپ را با دستور زیر فرمت کرده:

```
hadoop namenode -format
```


و در نهایت با دستور زیر هادوپ را اجرا می کنیم:

```
/usr/local/hadoop/sbin$ start-all.sh
```

برای اطمینان از صحت عملکرد آن دستور زیر را اجرا می کنیم:

```
/usr/local/hadoop/sbin$ jps
```

اگر خروجی تولید شده مانند زیر باشد، هادوپ به درستی اجرا شده است:

```
۹۰۲۶ NodeManager
```

```
۷۳۴۸ NameNode
```

```
۹۷۶۶ Jps
```

```
۸۸۸۷ ResourceManager
```

```
۷۵۰۷ DataNode
```

برای متوقف کردن هادوپ می توان از دستور زیر استفاده کرد:

```
/usr/local/hadoop/sbin$ stop-all.sh
```

۷-۲ اجرای مثال Word Count با هادوپ

یک فایل جاوا با نام WordCount ایجاد کرده و کد زیر را درون آن می نویسیم:

```
import java.io.IOException;
```

```
import java.util.StringTokenizer;
```

```
import org.apache.hadoop.conf.Configuration;
```

```
import org.apache.hadoop.fs.Path;
```

```
import org.apache.hadoop.io.IntWritable;
```

```
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Job;
```

```

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class WordCount {

    public static class TokenizerMapper extends Mapper<Object, Text, Text,
IntWritable>{

        private final static IntWritable one = new IntWritable(1);

        private Text word = new Text();

        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {

            StringTokenizer itr = new StringTokenizer(value.toString());

            while (itr.hasMoreTokens()) {

                word.set(itr.nextToken());

                context.write(word, one);

            }

        }

    }

}


    public static class IntSumReducer extends
Reducer<Text,IntWritable,Text,IntWritable> {

        private IntWritable result = new IntWritable();

```

```

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {

        int sum = 0;

        for (IntWritable val : values) {

            sum += val.get();

        }

        result.set(sum);

        context.write(key, result);

    }
}

```

```

public static void main(String[] args) throws Exception {

    Configuration conf = new Configuration();

    Job job = Job.getInstance(conf, "word count");

    job.setJarByClass(WordCount.class);

    job.setMapperClass(TokenizerMapper.class);

    job.setCombinerClass(IntSumReducer.class);

    job.setReducerClass(IntSumReducer.class);

    job.setOutputKeyClass(Text.class);

    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));

    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
}

```

```
}  
  
}
```

با دستور زیر آن را کامپایل کرده و class آن را می سازیم:

```
bin/hadoop com.sun.tools.javac.Main WordCount.java
```

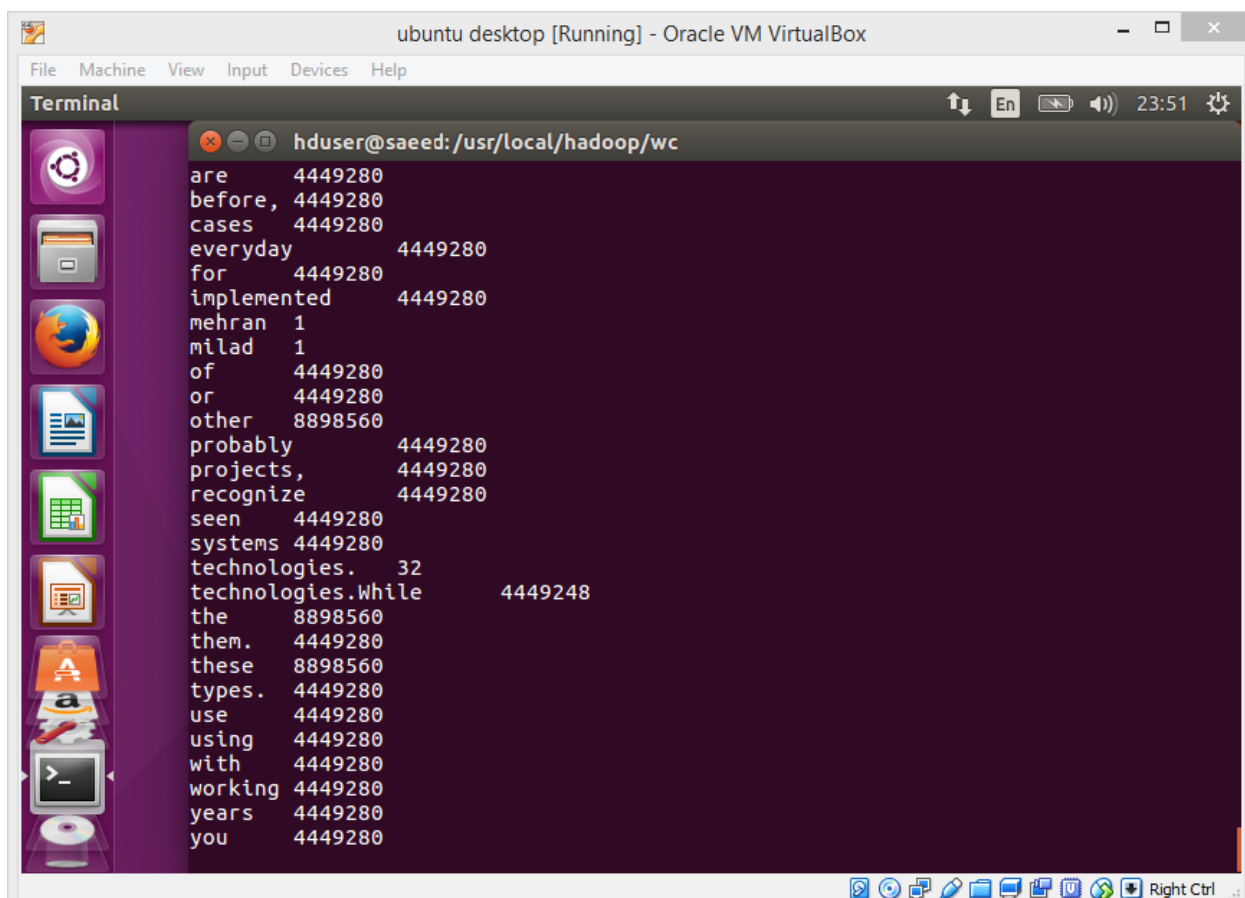
سپس با دستور زیر فایل jar آن را تولید می کنیم:

```
jar cf wc.jar WordCount*.class
```

پوشه ای با نام input می سازیم و تعدادی فایل متنی جهت شمارش تعداد کلمات آن درون آن قرار می دهیم.

```
bin/hadoop jar wc.jar WordCount /user/saeed/wordcount/output  
/user/saeed/wordcount/input
```

پس از اجرای دستور فوق، فایلی در پوشه output ایجاد شده و نتیجه شمارش را ذخیره کرده است.



شکل ۷-۱

۳-۷ استفاده از ماشین مجازی هدوپ

روش آسان برای داشتن هدوپ، استفاده از بسته های آماده هدوپ است که توسط شرکت های مختلفی تهیه شده اند. بسته های آماده هدوپ را شرکت های مختلفی چه به صورت بسته کامل و چه به صورت گزینشی ارائه می کنند. این شرکت ها عبارتند از:

IBM ♦

Cloudera ♦

Hortonworks ♦

هر کدام از این شرکت ها بسته های آماده ای را به صورت رایگان فراهم می کنند که هرچند دارای حجم بالایی هستند ولی می توانند توسط نرم افزارهای مجازی سازی به صورت مستقل اجرا شوند و همچنین این بسته ها عموماً دارای امکانات بالاتری نسبت به بسته عمومی هدوپ است. این بسته های آماده حتی شامل واسط های مدیریتی گرافیکی است که توانایی کار با هدوپ را آسانتر می کند. نکته دیگر این است که این بسته ها شامل زیرپروژه های بیشتری نسبت به بسته عمومی می باشند.

برای استفاده از این بسته ها کافی است به لینک های زیر مراجعه کنید و بسته مربوط به نرم افزار مجازی سازی خود را بارگیری نمایید و اجرا کنید:

♦ [بسته ماشین مجازی اختصاصی مرجع هدوپ ایران](#)

♦ [Cloudera QuickStart VMs](#)

♦ [InfoSphere BigInsights Quick Start Edition](#)

♦ [Hortonworks Data Platform](#)

به دلیل وجود تحریم ها و ممنوعیت استفاده ایرانیان، بسته ماشین مجازی مرجع هدوپ ایران را دریافت می کنیم و از آن برای اجرای مثال استفاده می کنیم.

برای دریافت این ماشین مجازی، ابتدا از طریق لینک زیر تصویر آن را دریافت می کنیم:

<https://hadoop.ir/box/>

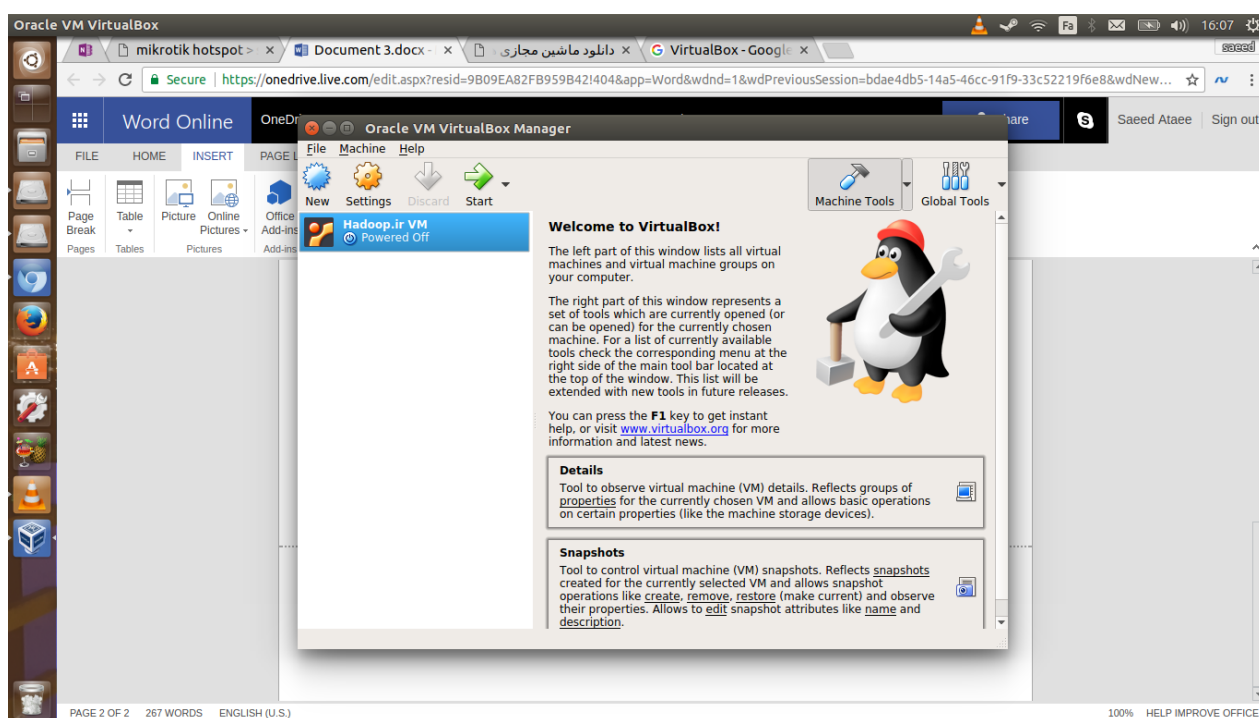
برای اجرای آن از VirtualBox استفاده می کنیم.



VirtualBox

شکل ۷-۲

پس از دانلود تصویر، طبق فیلم آموزشی موجود در سایت، تصویر را به VirtualBox اضافه می کنیم و آن را Start می کنیم.



شکل ۷-۳

پس از بالا آمدن کامل ماشین مجازی، دستور زیر را برای مشاهده لیست فایل های هدوپ می زنیم:

```
hadoop fs -ls
```

برای اجرای مثال شمارش کلمات به صورت زیر عمل می کنیم.

۱- ایجاد فایل mytest جهت قرار دادن فایل های مثال در آن با دستور زیر:

```
hadoop fs -mkdir mytest
```

۲- ایجاد فایل input برای قرارگیری فایل های ورودی در آن:

```
hadoop fs -mkdir mytest/input
```

۳- ایجاد فایل ورودی جهت تست با دستور زیر:

```
nano test.txt
```

در فایل وارد شده ورودی دلخواه را وارد می کنیم و آن را ذخیره می کنیم و می بندیم.

۴- فایل تست ایجاد شده را در به قسمت فایل های ورودی منتقل می کنیم:

```
hadoop fs -put test.txt mytest/input
```

۵- برای اجرای مثال به مسیر زیر می رویم:

```
cd /opt/hadoop/share/hadoop/mapreduce
```

۶- برای اجرای مثال دستور زیر را وارد می کنیم:

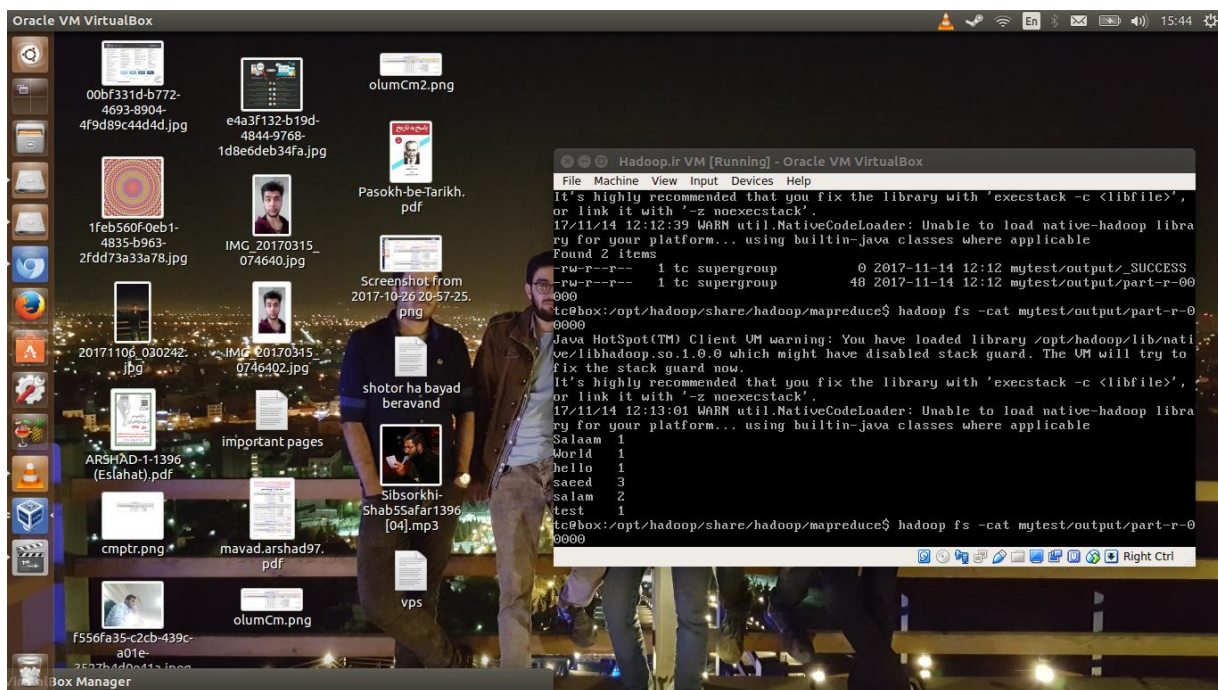
```
hadoop jar hadoop-mapreduce-examples-۲,۷,۱.jar wordcount mytest/input  
mytest/output
```

۷- پس از اجرای کامل دستور فوق، می توان خروجی ایجاد شده را در مسیر mytest/output مشاهده کرد:

hadoop fs -ls mytest/output

۸- فایل part-r-..... نتیجه را ذخیره کرده است.

hadoop fs -cat mytest/output/part-r-.....

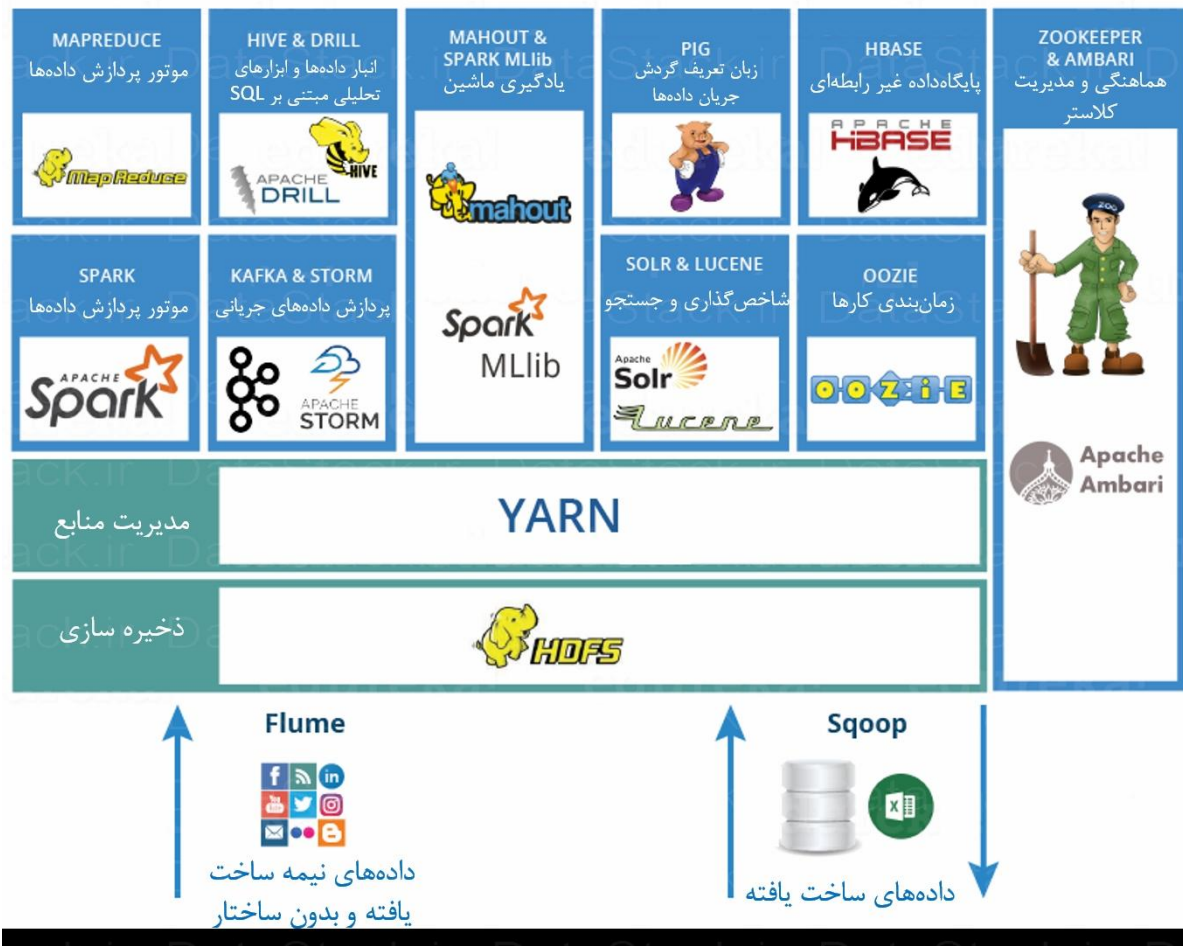


شکل ۴-۷

فصل ۸

اکوسیستم هدوپ

اکوسیستم هدوپ سکویی است که با مجموعه ابزارهایی که در اختیار دارد می‌تواند به حل مشکلات کلان داده‌ها کمک کند. هدوپ را می‌توان به عنوان مجموعه‌ای در نظر بگیرید که دربردارنده‌ی تعدادی از سرویس‌های (جمع‌آوری، ذخیره، تجزیه و تحلیل و نگهداری) کار با داده در درون خود می‌باشد. در ادامه بررسی مختصری از نحوه‌ی عملکرد سرویس‌های آن، هم به‌طور مجزا و هم در همکاری با یکدیگر می‌پردازیم که می‌توان گفت در مجموع اکوسیستم هدوپ را تشکیل می‌دهند:



شکل ۸-۱

۸-۱ سیستم فایل توزیع شده هدوپ - HDFS

سیستم فایل توزیع شده‌ی هدوپ که عنصر سازنده و زیربنایی هدوپ است و به عبارتی می‌توان از آن به عنوان ستون اصلی اکوسیستم هدوپ یاد کرد. این فایل سیستم امکان ذخیره‌سازی انواع مختلفی از مجموعه‌ی داده‌ها (داده‌های ساختاریافته، بدون ساختار و نیمه ساخت یافته) را فراهم می‌آورد. همچنین HDFS، مفهومی انتزاعی پیرامون منابع کلاستر ایجاد می‌کند، که این موضوع به ما این اجازه را می‌دهد تا آن را به عنوان یک واحد مجزا در نظر بگیریم. برای این منظور، HDFS داده‌ها

را در سرتاسر کلاستر در گره‌های مختلف و فراداده‌های داده را در یک گره نگهداری می‌کند. HDFS دو عنصر سازنده‌ی اصلی دارد: Namenode و Datanode

- ♦ Namenode گره‌ی اصلی در سیستم فایل HDFS است و فراداده‌های مربوط به داده‌ها را ذخیره می‌کند. درست مانند یک فایل ثبت رخداد ، دربر دارنده‌ی فراداده‌ها است و به عبارتی می‌توان آن را به عنوان جدول محتویات سیستم فایل تعبیر کرد. بنابراین، نیاز به فضای ذخیره‌سازی کمتر و منابع محاسباتی (مخصوصاً RAM) بالا دارد.
- ♦ از طرفی دیگر، تمام داده‌های شما در Datanodes ذخیره می‌گردد و به همین خاطر نیاز به منابع ذخیره‌سازی بیشتری دارد. این Datanode ها، سخت‌افزار ارزان قیمت در یک محیط توزیع شده هستند (شاید چیزی نظیر لپ‌تاپ‌ها و کامپیوترهای شخصی شما). به همین علت است که راهکارهای ارائه شده توسط هدوپ بسیار مقرون به صرفه هستند.



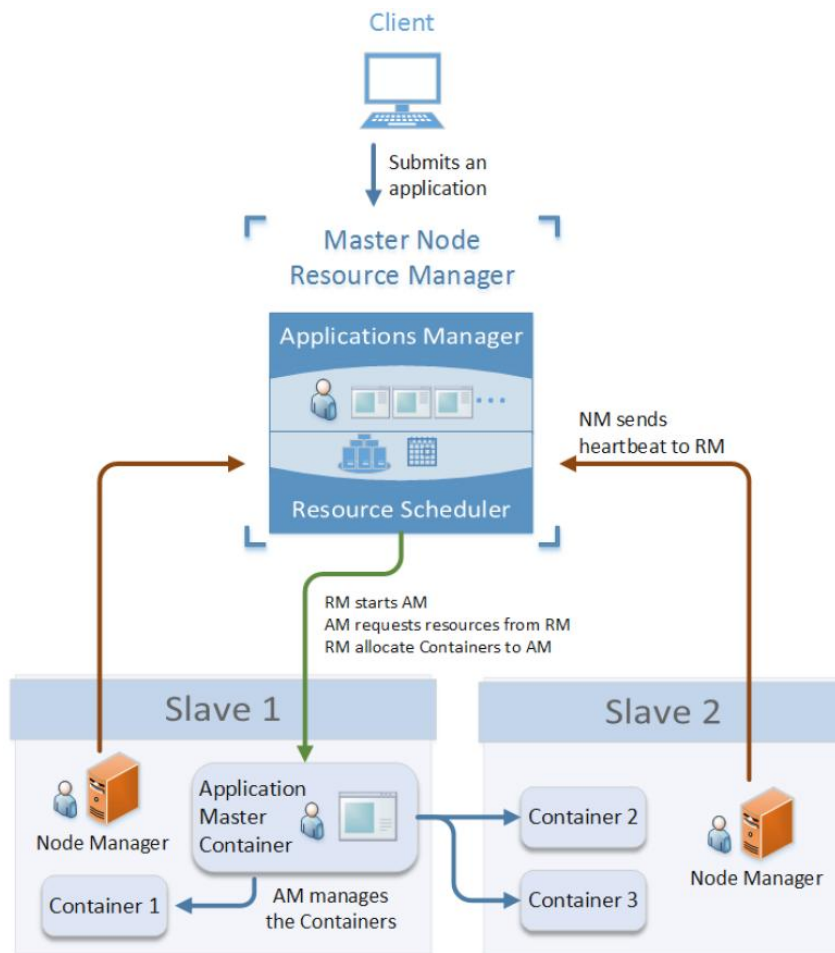
شکل ۸-۲

۸-۲ چهارچوب مدیریت منابع YARN

YARN را به عنوان مغز اکوسیستم هدوپ در نظر بگیرید. این ابزار تمامی فعالیت‌های پردازشی شما را به وسیله‌ی تخصیص منابع و برنامه‌ریزی، انجام می‌دهد. YARN دو جزء اصلی دارد: ResourceManager و NodeManager:

ResourceManager همان گرهی اصلی در بخش پردازش است.

- ♦ درخواست‌های پردازشی را دریافت می‌کند و سپس هر بخش از درخواست‌ها را به NodeManager متناسب با هر درخواست ارسال می‌کند (جایی که پردازش حقیقی رخ می‌دهد).
- ♦ NodeManager ها بر روی گره‌های Slave نصب می‌شوند. NodeManager ها مسئول اجرای وظیفه بر روی هر کدام از ماشین‌های Slave هستند.



شکل ۸-۳

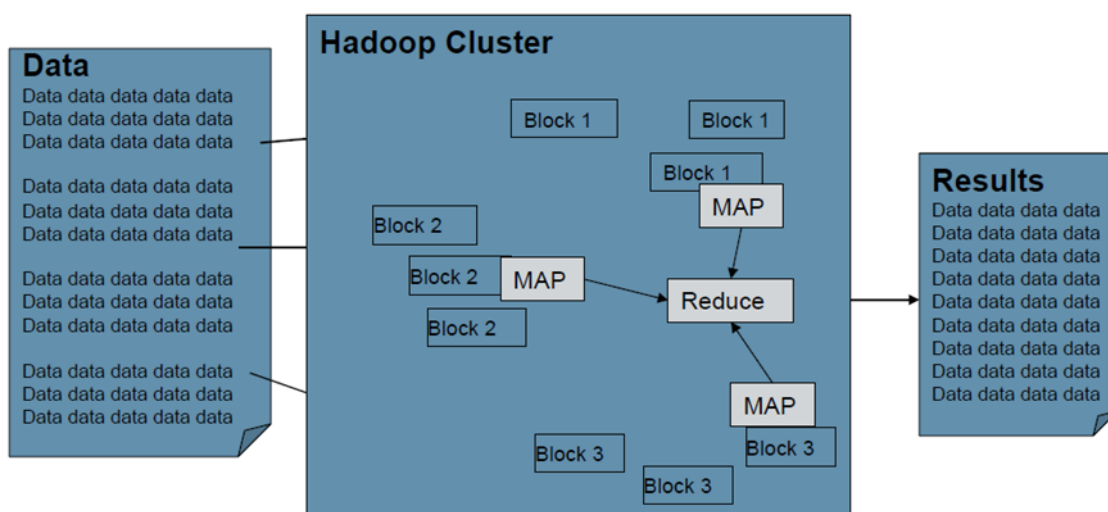
مولفه ResourceManager نیز از دو جزء تشکیل شده: Scheduler و ApplicationManager.

Scheduler بر اساس منابع مورد نیاز برنامه و براساس پیکربندی کلاستر، هنگام تخصیص منابع الگوریتم‌های زمان‌بندی را اجرا و براساس نتایج آن منابع را تخصیص می‌دهد. ApplicationManager نیز در هنگام قبول یک کار جدید، بررسی‌های لازم برای پذیرفته شدن کار در کلاستر را انجام می‌دهد. اینکه آیا منابع کلاستر پاسخگوی نیاز کار باشد، شماره

کار جدید قبلا در کلاستر نباشد و بررسی‌های دیگر را انجام داده و در صورت پذیرفته شدن کار، آن را به مولفه Scheduler تحویل می‌دهد.

۸-۳ موتور پردازش MapReduce

نگاشت/کاهش ابزار پردازش در اکوسیستم هدوپ است، که یک لایه پردازشی منطقی ارائه می‌دهد. به عبارت دیگر، نگاشت/کاهش یک چارچوب نرم‌افزاری است که به نوشتن برنامه‌هایی کمک می‌کند، که مجموعه‌ای داده‌های عظیم را با استفاده از الگوریتم‌های موازی و توزیع شده، در محیط هدوپ پردازش می‌کنند.



شکل ۸-۴

در یک برنامه‌ی نگاشت/کاهش، توابع Map() و Reduce()، دو وظیفه‌ی مجزا دارند:

- ♦ وظیفه‌ی Map فعالیت‌هایی هم‌چون فیلتر کردن، گروه‌سازی و ذخیره‌سازی است.
- ♦ در حالی که وظیفه‌ی Reduce جمع‌آوری و خلاصه‌سازی نتایجی است که از طریق Map حاصل شده است.
- ♦ در واقع نتایجی که از طریق وظیفه‌ی Map حاصل شده، یک جفتِ کلید/مقدار (K/V) است، که یک داده‌ی ورودی برای وظیفه‌ی Reduce محسوب می‌شود.

۸-۴ آپاچی ماهوت APACHE MAHOUT

ابزار پرکاربرد دیگر اکوسیستم هدوپ، آپاچی ماهوت می‌باشد. ماهوت واسطه‌هایی برای ایجاد برنامه‌های قابل مقیاس‌پذیر یادگیری ماشین فراهم می‌کند.



شکل ۸-۵

یادگیری ماشین چیست؟

الگوریتم‌های فراگیری ماشینی، اجازه‌ی ساخت سیستم‌های خود-یادگیرنده را به ما می‌دهند، که بدون آن که نیاز به برنامه‌ریزی مشخصی داشته باشد، تکامل می‌یابند. بر اساس رفتار کاربر، الگوهای داده و تجربیات گذشته می‌توانند در تصمیمات آینده یاری رسان باشند.

Mahout چه کاری انجام می‌دهد؟

پالایش گروهی، خوشه‌بندی و قواعد انجمنی کارهایی است که می‌توان توسط Mahout انجام داد:

♦ **پالایش گروهی:** Mahout رفتارهای کاربران، الگوهای آن‌ها و ویژگی‌هایشان را بررسی می‌کند و بر اساس آن پیش‌بینی‌هایی انجام می‌دهد و به کاربران پیشنهاد می‌دهد. رایج‌ترین کاربرد آن، استفاده در وب‌سایت‌های تجارت الکترونیکی است.

♦ **خوشه‌بندی:** خوشه‌بندی یا آنالیز خوشه در یادگیری ماشین، یکی از شاخه‌های یادگیری بی‌نظارت می‌باشد و فرآیندی است که در طی آن، نمونه‌ها به دسته‌هایی که اعضای آن مشابه یکدیگر می‌باشند تقسیم می‌شوند که به این دسته‌ها خوشه گفته می‌شود. بنابراین خوشه مجموعه‌ای از اشیاء می‌باشد که در آن اشیاء با یکدیگر مشابه بوده و با اشیاء موجود در خوشه‌های دیگر غیر مشابه می‌باشند.

♦ **طبقه‌بندی:** این روش به طبقه‌بندی و دسته‌بندی داده‌ها به زیر مجموعه‌هایی بر اساس خصوصیاتشان اشاره دارد. در واقع در این روش مجموعه‌ای از قوانین بر اساس داده‌های موجود (داده‌های مجموعه آموزش) ایجاد می‌شود تا بر اساس آن طبقه‌بندی مناسبی برای موضوع جدید در میان طبقه‌بندی‌های مختلف انجام گیرد. این روش جزء روش‌های یادگیری با نظارت محسوب می‌شود.

♦ **قواعد انجمنی:** قوانین انجمنی روابط و وابستگی‌های متقابل بین مجموعه بزرگی از اقلام داده‌ای را نشان می‌دهند. پیدا کردن چنین قوانینی می‌تواند در حوزه‌های مختلف مورد توجه بوده و کاربردهای متفاوتی داشته

باشد. بعنوان مثال کشف روابط انجمنی بین حجم عظیم تراکنش های کسب و کار می تواند در تشخیص تقلب، در حوزه پزشکی و همچنین در مورد اطلاعات روش بکارگیری وب توسط کاربران مورد استفاده قرار گیرد.

Mahout خط فرمانی برای فراخوانی الگوریتم های مختلف ارائه می دهد. این ابزار از مجموعه کتابخانه از پیش تعیین شده ای بهره می برد که در حال حاضر شامل الگوریتم های داخلی متعدد، برای کاربردهای مختلف است.

۵-۸ آپاچی زو کیپر – APACHE ZOOKEEPER

Apache Zookeeper یک ابزار هماهنگ کننده برای سرویس های در هدوپ است که برای اجرا نیاز به ترکیبی از سرویس های مختلف در اکوسیستم هدوپ دارند. Apache Zookeeper نقش یک هماهنگ کننده سرویس های مختلف در یک محیط توزیع شده را برعهده دارد.



شکل ۶-۸

قبل از عرضه ی Zookeeper، هماهنگی میان سرویس های مختلف در اکوسیستم هدوپ بسیار دشوار و زمان بر بود. پیش از آن، سرویس ها مشکلات زیادی برای تعامل با مواردی همچون پیکربندی کردن سیستم ها همراه با همگام نگه داشتن داده ها داشتند. حتی اگر سرویس ها پیکربندی می شدند، تغییر در پیکربندی سرویس ها تبدیل به مسئله ای پیچیده می شد و مدیریت آن را دشوار می کرد.

Zookeeper برای حل مشکلات بالا خلق شد. با انجام وظایفی نظیر همگام سازی، حفظ پیکربندی، گروه بندی و نام گذاری، به طور چشم گیری در زمان حل اینگونه مسائل صرفه جویی شد. برخلاف ساده بودن این سرویس ها، می توان از آن برای ارائه ی راه حل های قدرتمند استفاده کرد. شرکت های بزرگی همچون Yahoo، Rackspace و eBay از این سرویس در بسیاری از موارد کاری خود بهره می برند، در نتیجه می توان به ارزش و اهمیت این ابزار پی برد.

فصل ۹

آپاچی هایو

Apache Hive

۹-۱ معرفی

فیسبوک، Hive را برای آن دسته از افرادی طراحی کرده که می‌توانند به راحتی با SQL کار کنند. در نتیجه، شما با تجربه‌ای نظیر کار با دستورات SQL مشغول کار کردن در اکوسیستم هادوپ هستید. در واقع، Hive یک انبار داده در اکوسیستم هادوپ است که مسئولیت خواندن، نوشتن و مدیریت مجموعه داده‌های بزرگ را در یک محیط توزیع شده و با استفاده از واسطی مانند SQL، برعهده دارد.



شکل ۹-۱

هایو همه نوع داده‌های اولیه‌ی SQL را پشتیبانی می‌کند. همچنین می‌توانید برای انجام دادن نیازهای خاص خود، از تابع‌های نوشته شده توسط کاربر (UDF) و یا توابع از پیش تعیین شده، استفاده کنید.

زبان پرس‌وجوی Hive، Hive Query Language یا HQL نامیده می‌شود، که بسیار مشابه زبان SQL است.

$$\text{SQL} = \text{HQL} + \text{Hive}$$

این زبان از دو جزء اصلی تشکیل شده: خط فرمان Hive و درایور JDBC/ODBC.

- ♦ رابط خط فرمان Hive، که برای اجرای دستورات HQL مورد استفاده قرار می‌گیرد.
- ♦ در حالی که درایور پایگاه داده‌ی جاوا (JDBC) و درایور پایگاه داده‌ی اشیاء (ODBC)، به منظور ایجاد اتصالی از ذخیره‌سازهای داده استفاده می‌شود.

۲-۹ نصب آپاچی Hive بر روی اوبونتو

برای نصب، ورژن مناسب آن را از [سایت آپاچی](http://www-us.apache.org/dist/hive/stable/) دانلود و سپس مراحل زیر را طی می کنیم

۱- مراجعه به آدرس <http://www-us.apache.org/dist/hive/stable/> و دریافت فایل [apache-hive-۱,۲,۲-bin.tar.gz](http://www-us.apache.org/dist/hive/stable/apache-hive-۱,۲,۲-bin.tar.gz)

۲- خارج کردن فایل از حالت فشرده با دستور مقابل

```
> tar -xzf apache-hive-۱,۲,۲-bin.tar.gz
```

۳- تمامی فایل ها را به مسیر مناسبی منتقل می کنیم

```
> mkdir /usr/local/hive
```

```
> mv apache-hive-۱,۲,۲-bin/* /usr/local/hive/
```

۴- فایل `bashrc` را تغییر داده و کدهای زیر را به انتهای آن اضافه می کنیم

```
# HIVE HOME PATH
```

```
export HIVE_HOME=/usr/local/hive
```

```
export PATH=$PATH:$HIVE_HOME/bin
```

```
# END HIVE HOME PATH
```

۵- دایرکتوری مربوط به هایو را در `hdfs` می سازیم

```
> hadoop fs -mkdir /usr/hive/warehouse
```

این دایرکتوری برای ذخیره کردن جدول ها و داده های مربوط به هایو خواهد بود.

۶- پوشه `tmp` را به منظور نگهداری نتایج میانی پردازش می سازیم

```
> hadoop fs -mkdir /usr/tmp
```

۷- دسترسی های لازم برای تغییر و نوشتن این فایل ها را ایجاد می کنیم

```
> hadoop fs -chmod g+w /usr/hive/warehouse
```

```
> hadoop fs -chmod g+w /usr/tmp
```

۸- به فایل hive-conf.sh خط زیر را اضافه می کنیم

```
> nano /usr/local/hive/bin/hive-conf.sh
```

```
export HADOOP_HOME=/usr/local/hadoop
```

۹- برای اجرای hive و دسترسی به خط فرمان آن از دستور زیر استفاده می کنیم

```
> hive
```

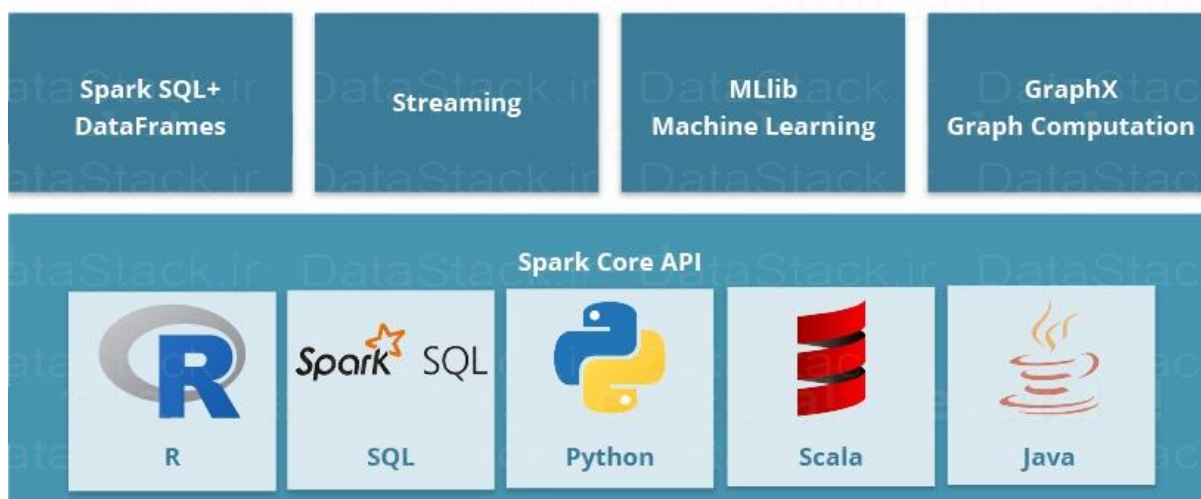
فصل ۱۰

آپاچی اسپارک APACHE SPARK

۱-۱۰ معرفی

آپاچی اسپارک چهارچوبی برای تجزیه و تحلیل داده‌های دسته‌ای و جریانی در یک محیط محاسباتی توزیع شده است. این ابزار به زبان برنامه‌نویسی اسکالا نوشته شده و نخست در دانشگاه برکلی کالیفرنیا توسعه یافته است. اسپارک از محاسبات درون حافظه‌ای به منظور افزایش سرعت پردازش داده‌ها استفاده می‌کند که در مقایسه با مدل نگاشت/کاهش از کارایی بهتری برخوردار است.

سرعت اسپارک برای پردازش داده‌های بزرگ با بهره‌وری از محاسبات درون حافظه‌ای و دیگر بهینه‌سازی‌ها، نزدیک به ۱۰۰ برابر سریع‌تر از هودپ در زمانیکه داده‌ها درون حافظه جای بگیرند و تا ده‌ها برابر نسبت به زمانی که تمام داده‌ها در حافظه قرار نگیرند. بنابراین، در مقایسه با MapReduce، قدرت پردازشی بالاتری دارد.



شکل ۱-۱۰

همان‌طور که مشاهده می‌کنید، اسپارک با کتابخانه‌های سطح بالایی ارائه شده است و زبان‌های برنامه‌نویسی R، Python، Scala و Java زبان‌هایی هستند که می‌توان توسط آن از این ابزار استفاده کرد. ابزار آپاچی اسپارک به مجموعه سرویس‌های مختلف، اجازه‌ی ادغام نمونه‌هایی هم‌چون کتابخانه‌های یادگیری ماشین MLlib، واسط‌های تحلیل داده‌های گرافی GraphX، واسط‌های تحلیل داده‌های رابطه‌ای SQL + Data Frames و سرویس‌های داده‌های جریانی، برای افزایش قابلیت‌های خودش می‌دهد.

۱۰-۲ نصب آپاچی اسپارک بر روی اوبونتو



شکل ۱۰-۲

۱- جاوا

برای اجرا و نصب اسپارک باید جاوا بر روی ماشین نصب باشد. با استفاده از دستور زیر چک می کنیم جاوا بر روی سیستم نصب هست یا خیر:

`Java -version`

در صورتی که جاوا نصب نباشد، با دستورات زیر اقدام به نصب آن کنید:

`sudo apt-get update`

`sudo apt-get install default-jdk`

```
root@saeed-X550CC:~# java -version
openjdk version "1.8.0_151"
OpenJDK Runtime Environment (build 1.8.0_151-8u151-b12-0ubuntu0.16.04.2-b12)
OpenJDK 64-Bit Server VM (build 25.151-b12, mixed mode)
root@saeed-X550CC:~#
```

شکل ۱۰-۳

۲- scala

پس از نصب جاوا بر روی ماشین، اقدام به نصب scala از طریق دستور زیر می نماییم:

`sudo apt-get install scala`

برای تست صحت تست دستور scala را برای اجرای آن در ترمینال وارد می کنیم و دستور زیر را وارد می کنیم:

```
println("Hello World")
```

و با دستور زیر، اجرای آن را خاتمه می‌دهیم:

```
:q
```

```
root@saeed-X550CC:~/spark-2.2.0-bin-hadoop2.7/bin# scala
Welcome to Scala version 2.11.6 (OpenJDK 64-Bit Server VM, Java 1.8.0_151).
Type in expressions to have them evaluated.
Type :help for more information.

scala> println("Hello World!")
Hello World!

scala> :q
root@saeed-X550CC:~/spark-2.2.0-bin-hadoop2.7/bin#
```

شکل ۱۰-۴

۳- گیت

برای نصب اسپارک، ابتدا باید ابزار گیت نصب شود. پس از دستور زیر برای نصب آن استفاده می‌کنیم:

```
sudo apt-get install git
```

۴- اسپارک

پس از نصب کامل آن، به صفحه دانلود آپاچی اسپارک به آدرس زیر رفته و ورژن مناسب را دریافت می‌کنیم:

<https://spark.apache.org/downloads.html>

پس از اتمام دانلود، فایل را از حالت فشرده خارج می‌کنیم:

```
tar xvf spark-۲,۰,۲-bin-hadoop۲,۷.tgz
```

سپس به دایرکتوری `bin` واقع در اسپارک رفته و فایل زیر را اجرا می‌کنیم:

```
cd /spark-۲,۲,۰-bin-hadoop۲,۷/bin
```

```
./spark-shell
```

با اجرای دستور زیر و گرفتن خروجی، از نصب کامل آن بر روی ماشین اطمینان حاصل می‌نماییم:

```
println("Spark shell is running")
```

```

root@saeed-X550CC:~/spark-2.2.0-bin-hadoop2.7/bin# ./spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/11/29 09:45:14 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/11/29 09:45:14 WARN Utils: Your hostname, saeed-X550CC resolves to a loopback address: 127.0.1.1; using 172.24.84.133 instead (on interface wlp3s0)
17/11/29 09:45:14 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/11/29 09:45:43 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0
17/11/29 09:45:44 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
17/11/29 09:45:49 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://172.24.84.133:4040
Spark context available as 'sc' (master = local[*], app id = local-1511948716000).
Spark session available as 'spark'.
Welcome to

  ____      __
 / ___ |__  / /_
/ /___|  \| / __ \|
 \___ \____/ \___ \
              |___|

version 2.2.0

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala> println("Spark shell is Running")
Spark shell is Running

scala>

```

شکل ۱۰-۵

۱۰-۳ اجرای مثال WordCount

پس از نصب اسپارک و اجرای آن، مثال شمارش کلمات را اجرا می کنیم. بدین منظور، در shell دستورات زیر را به ترتیب اجرا می کنیم:

```

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext._

val txtFile = "inputFile.txt"

val txtData = sc.textFile(txtFile)

txtData.cache()

```

برای مشاهده تعداد خط موجود در فایل می توان از دستور زیر استفاده کرد:

```
txtData.count()
```

برای نمایش تعداد کلمات موجود در آن از دستورات زیر استفاده می کنیم:

```

val wcData = txtData.flatMap(l => l.split(" ")).map(word => (word, ۱)).reduceByKey(_ + _)

wcData.collect().foreach(println)

```


خروجی به صورت زیر خواهد بود:

```
saeed@saeed-X550CC: ~  
(A,1)  
(through,1)  
(#,1)  
(library,1)  
(following,2)  
(More,1)  
(which,2)  
(also,4)  
(storage,1)  
(should,2)  
(To,2)  
(for,12)  
(Once,1)  
(["Useful",1)  
(setup,1)  
(mesos://,1)  
(Maven](http://maven.apache.org/),1)  
(latest,1)  
(processing,,1)  
(the,24)  
(your,1)  
(not,1)  
(different,1)  
(distributions.,1)  
(given.,1)  
(About,1)  
(if,4)  
(instructions.,1)  
(be,2)  
(do,2)  
(Tests,1)  
(no,1)  
(project.,1)  
(./bin/run-example,2)  
(programs.,1)  
(including,4)  
(./bin/run-example,1)  
(Spark.,1)  
(Versions,1)  
(started,1)  
(HDFS,1)  
(by,1)  
(individual,1)
```

شکل ۶-۱۰

آپاچی اچ بیس

APACHE HBASE



شکل ۱۱-۱

۱۱-۱ معرفی

HBase پایگاه داده‌ای توزیع شده، غیر رابطه‌ای، متن‌باز و یکی از انواع پایگاه داده‌های NoSQL محسوب می‌شود. این پایگاه داده به زبان جاوا نوشته شده و از نوع سطر گسترده می‌باشد. ایده اولیه آن از محصول BigTable گوگل مدل‌سازی شده است، که برای سیستم‌های ذخیره‌سازی توزیع شده طراحی شده تا بتواند از عهده‌ی مجموعه داده‌های بزرگ برآید. HBase برای زیرساخت خود از HDFS استفاده می‌کند و قابلیت‌هایی نظیر BigTable را فراهم می‌سازد. به منظور ارتباط با این پایگاه داده علاوه بر کلاینت‌های مختلف که برای اکثر زبان‌های برنامه‌نویسی توسعه داده شده اند، می‌توان از مکانیسم‌های REST, Avro و Thrift APIs نیز استفاده کرد.

۱۱-۲ نصب آپاچی hbase بر روی اوبونتو

نصب hbase به سه صورت Standalone و Pseudo-Distributed و Fully Distributed امکان پذیر است. نصب در حالت اول هیچ وابستگی به هدوپ ندارد و حالت پیش فرض است و بر روی یک jvm اجرا می‌شود. حالت شبه توزیع شده آن از سیستم هدوپ تگ گره استفاده می‌کند و بر روی HDFS هدوپ اجرا می‌شود. حالت توزیع شده آن بر روی هدوپ به صورت چند گره اجرا می‌شود و به شدت برای محیط production توصیه می‌شود.

در اینجا ما به نصب حالت اول آن می‌پردازیم. بدین منظور، ابتدا ورژن مناسب آن را از [سایت آپاچی](http://www-eu.apache.org/dist/hbase/stable/) دانلود می‌کنیم.

۱- مراجعه به آدرس <http://www-eu.apache.org/dist/hbase/stable/> و دریافت فایل [hbase-۱,۲,۶-bin.tar.gz](#)

[bin.tar.gz](#)

۲- خارج کردن فایل از حالت فشرده با دستور زیر

```
> tar xvfz hbase-۱,۲,۶-bin.tar.gz
```

۳-انتقال تمامی فایل ها به مسیر مناسب برای اجرای آن

```
> mv hbase-۱,۲,۶/* /usr/local/hbase/
```

۴- تغییر فایل hbase-env.sh و اصلاح مسیر جاوا در آن

```
> nano hbase-env.sh
```

و تغییر آن به حالت زیر JAVA_HOME یافتن مقدار

```
> export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")
```

۵- فایل `bashrc` نیز باید تغییر کند و آدرس `hbase` را در آن درج کنیم

```
> nano ~/.bashrc
```

و موارد زیر را به انتهای آن اضافه می کنیم

```
# HBASE PATH
```

```
export HBASE_HOME=/usr/local/hbase
```

```
export PATH=$PATH:$HBASE_HOME/bin
```

و از دستور زیر برای اعمال تغییرات در سیستم استفاده می کنیم

```
> source ~/.bashrc
```

۶- فایل `hbase-site.xml` باید به صورت زیر تغییر کند

```
> mkdir -p /usr/local/hbase_store/hbase
```

```
> mkdir -p /usr/local/hbase_store/zookeeper
```

```
> nano hbase-site.xml
```

و محتوای فایل را به صورت زیر تغییر می دهیم:

```
<configuration>
```

```
<property>
```

```
<name>hbase.rootdir</name>
```

```
<value>file:///usr/local/hbase_store/hbase</value>
```

```
</property>
```

```
<property>
```

```
<name>hbase.zookeeper.property.dataDir</name>
```

```
<value>/usr/local/hbase_store/zookeeper</value>
```

```
</property>
```

```
</configuration>
```

۷- در گام بعدی چک می کنیم موارد زیر حتما در فایل `etc/hosts` به صورت زیر وجود داشته باشد

۱۲۷.۰.۰.۱ localhost

۱۲۷.۰.۰.۱ saeed-X550CC

۸- اکنون `hbase` را با دستور زیر اجرا می کنیم

```
> cd /usr/local/hbase/bin
```

```
> start-hbase.sh
```

پس از اجرا شدن دستور فوق، با زدن دستور `jps` می توان از صحت اجرای آن اطمینان حاصل کرد

```
root@saeed-X550CC:/usr/local/hbase/bin# start-hbase.sh
starting master, logging to /usr/local/hbase/logs/hbase--master-saeed-X550CC.out
OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
root@saeed-X550CC:/usr/local/hbase/bin# jps
348 Jps
94 HMaster
root@saeed-X550CC:/usr/local/hbase/bin#
```

شکل ۱۱-۲

۹- اجرای `shell` مربوط به `hbase` از طریق دستور زیر امکان پذیر است که امکان اجرای دستورات مختلف به `hbase` را

برای ما مقدر می کند

```
> hbase shell
```

```
root@saeed-X550CC:/usr/local/hbase/bin# hbase shell
2017-12-04 09:58:04,447 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load

hbase(main):002:0>
```

شکل ۱۱-۳

منابع و مآخذ

- ۱- وبسایت فارسی داکر - Docker.ir
- ۲- مخزن ایرانی تصاویر داکر - Elastic.io
- ۳- وبسایت رسمی آپاچی - Apache.org
 - a. <https://hadoop.apache.org/>
 - b. <https://hive.apache.org/>
 - c. <http://spark.apache.org/>
 - d. <http://hbase.apache.org/>
 - e. <http://mahout.apache.org/>
 - f. <http://zookeeper.apache.org>
- ۴- ویکی پدیا
 - a. https://en.wikipedia.org/wiki/Big_data
 - b. [https://en.wikipedia.org/wiki/Docker_\(software\)](https://en.wikipedia.org/wiki/Docker_(software))
 - c. https://en.wikipedia.org/wiki/Apache_hive
 - d. https://en.wikipedia.org/wiki/Apache_Spark
- ۵- <https://www.ibm.com/developerworks/library/bd-yarn-intro>
- ۶- <https://www.infoq.com/articles/apache-spark-introduction>
- ۷- مرجع آموزش و ارزیابی زبان‌های برنامه نویسی - Tutiran.com
- ۸- مرجع هدوپ ایران - Hadoop.ir
- ۹- وبسایت دیتا استک - Datastack.ir
- ۱۰- <http://www.guru۹۹.com/hbase-installation-guide.html>
- ۱۱- <http://www.guru۹۹.com/hbase-limitations-advantage-problems.html>
- ۱۲- http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_single_node_cluster.php
- ۱۳- کتاب تحلیل‌های عظیم داده: نقشه راه پیاده سازی، فناوری و ابزارها - دیوید لوشین - ترجمه سعید روحانی، سمیه حسینی
- ۱۴- کتاب اصول داده‌های بزرگ: مفاهیم، پیشران‌ها و تکنیک‌ها - توماس ارل - ترجمه حسن رشیدی