دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

پایان‌نامه کارشناسی

گرایش نرم‌افزار

عنوان

پیاده‌سازی یک ابزار داده‌کاوی مبتنی بر آپاچی اسپارک برای داده‌های جاری

نگارش

سینا شیخ‌الاسلامی

اساتید راهنما

دکتر امیرحسین پی‌براه

دکتر سید رسول موسوی

تیرماه ۱۳۹۵

اینجانب سینا شیخ‌الاسلامی متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است، مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

سینا شیخ‌الاسلامی

امضا

ای خدا ای فضل تو حاجت روا

با تو یاد هیچ کس نبود روا

این قدر ارشاد تو بخشیده‌ای

تا بدین بس عیب ما پوشیده‌ای

قطره‌ی دانش که بخشیدی ز پیش

متصل گردان به دریاهای خویش

قطره‌ی علم است اندر جان من

وا رهانش از هوا و ز خاک تن

«مولانا جلال‌الدین محمد بلخی»

تقدیر و تشکر:

بر خود لازم می‌دانم تا از بزرگوارانی که در این مسیر مرا یاری کرده‌اند سپاس‌گزاری کنم، هرچند این قلم تاب بیان قدر و منزلتشان را ندارد:

از مادر مهربان، پدر قهرمان، و خواهر عزیزتر از جانم، که هرچه دارم از آن‌هاست،

از امیرحسین پی‌براه عزیز، استاد ارجمند و دوست ارزشمندم، که به من در انتخاب مسیر آینده‌ام کمک کرد، راهنمای من در انجام این پروژه بود و ورای همه‌ی این‌ها، امید را در دل من و دوستانم زنده نگاه داشت و من خود را همیشه شاگرد ایشان خواهم دانست،

از جناب آقای دکتر سید رسول موسوی، که پشتیبان، حامی و راهنمای من در حین انجام این پروژه بودند،

از جناب آقای دکتر علیرضا باقری، که علاوه بر قبول زحمت داوری این پروژه، در طول این مسیر از راهنمایی‌هایشان بهره‌ی بسیار بردم،

و سرانجام، از دوستان عزیزم: فرزاد نوذریان، ملیحه هاشمی، ساسان دلیر، ریحانه شاه‌محمدی، کیوان ساسانی، آرمین باشی‌زاده، محمد قریشی، محمد احمدپناه، و پویا پارسا، که یاری و حمایتشان اندازه نداشت.

چکیده

کاوش و پردازش داده‌های جاری همواره بخش مهمی از پژوهش‌های مربوط به داده‌کاوی را به خود اختصاص داده است. با این وجود، با پیشرفت‌های اخیر در فناوری‌های رایانش ابری و سیستم‌های توزیع شده و همچنین فراگیر شدن استفاده از روش‌ها و ابزارهای تحلیل داده‌های حجیم، و به دلیل چالش‌ها و ویژگی‌های منحصر به فرد این دسته از داده‌ها، پژوهش در زمینه‌ی کاوش داده‌های جاری اهمیت روزافزونی یافته است.

در سالیان اخیر بسترهای مختلفی برای پردازش داده‌های حجیم و جریان‌داده‌ها تولید شده‌اند که از میان آن‌ها می‌توان به آپاچی اسپارک، آپاچی استورم، و آپاچی فلینک اشاره کرد. این بسترها دارای ابزارهایی برای پردازش داده‌های جاری هستند اما هنوز بسیاری از الگوریتم‌ها و روش‌های متداول کاوش داده‌های جاری برای استفاده برروی این بسترها پیاده‌سازی و آماده نشده‌اند.

هدف از این پروژه، پیاده‌سازی ابزاری متن‌باز برای کاوش داده‌های جاری می‌باشد. این ابزار شامل کتابخانه‌ای از الگوریتم‌های کاوش و پردازش داده‌های جاری، رابط کاربری گرافیکی برای مدیریت منابع جریان‌داده، تعریف و اجرای عملیات داده‌کاوی، نمایش نتایج حاصل از اجرا، مدیریت و نظارت بر محیط اجرای عملیات، و همچنین یک تولیدکننده‌ی جریان‌داده می‌باشد. این ابزار بر بستر آپاچی اسپارک و رابط برنامه‌نویسی اسپارک‌استریمینگ به عنوان یکی از بسترهای پیشرو برای پردازش داده‌های جاری و به طور کلی داده‌های حجیم پیاده‌سازی شده است.

در این پایان‌نامه، در ابتدا توضیحاتی در مورد چرایی نیاز به پردازش و کاوش داده‌های جاری، و چالش‌ها و مفاهیم کلیدی مرتبط با پردازش داده‌های جاری ارائه خواهد شد. سپس، راه‌کارهای موجود برای حل این مسأله و راه حل پیشنهادی مورد بحث قرار خواهد گرفت. در نهایت، به شرح پیاده‌سازی راه حل، نتایج حاصل شده، و کارهای آینده پرداخته می‌شود.

واژه‌های کلیدی:

داده‌های جاری، داده‌کاوی، جریان‌داده‌ها، داده‌های حجیم، الگوریتم‌های توزیع شده، آپاچی اسپارک

فهرست عنوان‌ها

صفحه

فهرست شکل‌ها

صفحه

فهرست جدول‌ها

صفحه

فصل اول -

مقدمه

پیشرفت‌های اخیر در حوزه‌ی سخت‌افزار منجر به این شده است که جمع‌آوری پیوسته‌ی داده‌ها به کاری آسان و متداول تبدیل شود [۱][۲][۳]. کارهای روزانه‌ای مانند جستجو در وب، ارسال پست در شبکه‌های اجتماعی، و خرید از فروشگاه‌های اینترنتی، به مرور حجم زیادی از داده تولید می‌کنند و با پردازش و کاوش[۱] این داده‌ها می‌توان به نتایج جالبی دست پیدا کرد. به عنوان مثالی دیگر، سامانه‌های کنترل خطوط حمل و نقل و ترافیک به طور معمول با جریان عظیمی از داده‌ها روبه‌رو هستند که تحلیل سریع آن‌ها می‌تواند در تصمیم‌گیری به مسئولین این حوزه‌ها کمک شایانی کند. همچنین، با تحلیل و کاوش کم-تأخیر[۲] داده‌های مربوط به بسته‌های رد و بدل شده در یک شبکه‌ی کامپیوتری می‌توان به بروز ناهنجاری[۳] یا وقوع حملات خرابکارانه پی‌برد. در تمامی مثال‌های فوق، نوع خاصی از داده‌ها به نام داده‌های جاری[۴] مطرح هستند.

داده‌های جاری در مقایسه با دیگر انواع داده‌ها دارای خصوصیات منحصر به فردی هستند که پردازش و کاوش آن‌ها را به امری چالش‌برانگیز تبدیل می‌کند. از جمله‌ی این خصوصیات و چالش‌ها می‌توان به نیاز به الگوریتم‌های تک‌عبوره[۵]، نیاز به پردازش و کاوش بهنگام[۶] یا کم‌تأخیر، عدم امکان ذخیره‌ی همه‌ی داده‌ها برروی حافظه‌های انبوه و پایگاه داده‌ها، امکان تغییر در نرخ ورود و حجم داده‌ها، و وقوع تحول در داده‌ها اشاره کرد [۱][۲].

در سالیان اخیر بسترهای مختلفی برای پردازش داده‌های حجیم ایجاد شده و توسعه یافته‌اند که از پردازش داده‌های جاری هم پشتیبانی می‌کنند. از جمله‌ی این بسترها می‌توان به آپاچی اسپارک، آپاچی

---

[۱] Data Mining

[۲] Low-Latency

[۳] Anomaly

[۴] Stream Data

[۵] Single-pass Algorithms

[۶] Realtime

استورم، و آپاچی فلینک اشاره کرد. با این حال و با وجود این که این بسترها دارای ابزارها و قابلیت‌هایی برای پردازش به‌صرفه‌ی جریان‌داده‌ها هستند، کمبود کتابخانه‌های حاوی الگوریتم‌های معمول کاوش داده‌های جاری برای استفاده در این بسترها به چشم می‌خورد. از طرف دیگر، استفاده و بهره‌گیری از امکانات و قابلیت‌های این بسترها نیازمند دانش و تجربه‌ی فراوان در حوزه‌های مختلفی از جمله رایانش ابری[۷]، سیستم‌های توزیع‌شده[۸]، الگوریتم‌های موازی[۹]، و داده‌کاوی می‌باشد.

هدف از این پروژه، طراحی و پیاده‌سازی ابزاری مبتنی بر بسترهای توزیع‌شده پردازش داده‌های حجیم[۱۰] برای کاوش داده‌های جاری است. این ابزار که SDMiner نام دارد، شامل:

- یک رابط کاربری گرافیکی برای تعریف کارهای[۱۱] داده‌کاوی، مدیریت جریان‌داده[۱۲]های ورودی، و نمایش نتایج به کاربران، و
- کتابخانه‌ای از الگوریتم‌های کاوش و پردازش داده‌های جاری، مانند نمونه‌برداری تصادفی

می‌باشد. از میان بسترهای مختلف پردازش داده‌های حجیم، بستر توزیع‌شده آپاچی اسپارک[۱۳] برای استفاده‌ی این ابزار انتخاب شده است.

در ادامه‌ی این پایان‌نامه و در فصل دوم، به چالش‌ها، روش‌ها و ابزارهای پردازش داده‌های جاری پرداخته خواهد شد. فصل سوم به موازی‌سازی و پیاده‌سازی الگوریتم نمونه‌برداری تصادفی بدون تبعیض به عنوان یکی از معمول‌ترین الگوریتم‌های کاوش داده‌های جاری می‌پردازد. در فصل چهارم طراحی و پیاده‌سازی ابزار SDMiner مورد بررسی قرار خواهد گرفت. بدین منظور، معماری کلی و توصیف اجزای مختلف سیستم، متدولوژی مهندسی نرم‌افزار به کار رفته در طراحی و پیاده‌سازی این پروژه، جزئیات

---

[۷] Cloud Computing

[۸] Distributed Systems

[۹] Parallel Algorithms

[۱۰] Big Data

[۱۱] Job

[۱۲] Data Stream

[۱۳] Apache Spark

پیاده‌سازی قسمت‌های مختلف ابزار، و نتایج حاصل از پیاده‌سازی بیان خواهد شد. در نهایت، فصل پنج به جمع‌بندی و کارهای آینده مرتبط با این پروژه خواهد پرداخت.

فصل دوم –

چالش‌ها، روش‌ها و ابزارهای پردازش و کاوش داده‌های جاری

در این فصل، مفاهیم پایه‌ی مطرح در پروژه، از جمله خصوصیات داده‌های جاری، چالش‌های پردازش و کاوش آن‌ها، و راه‌کارهای موجود مورد بررسی قرار خواهد گرفت.

۲.۱ داده‌های جاری و کاربردهای آن‌ها

پیشرفت‌های سخت‌افزاری در سالیان اخیر منجر به این شده است که جمع‌آوری پیوسته‌ی داده‌ها به کاری آسان و متداول تبدیل شود. کارهای روزانه‌ای مانند جستجو در وب، ارسال پست در شبکه‌های اجتماعی، و خرید از طریق فروشگاه‌های اینترنتی، به مرور حجم زیادی از داده تولید می‌کنند و با پردازش و کاوش این داده‌ها می‌توان به نتایج جالبی دست پیدا کرد. به عنوان مثالی دیگر، سامانه‌های کنترل خطوط حمل و نقل و ترافیک به طور معمول با جریان عظیمی از داده‌ها روبه‌رو هستند که تحلیل سریع آن‌ها می‌تواند به مسئولین در تصمیم‌گیری‌ها کمک کند. همچنین، با تحلیل و کاوش کم-تأخیر داده‌های مربوط به بسته‌های رد و بدل شده در یک شبکه‌ی کامپیوتری، می‌توان به بروز ناهنجاری یا وقوع حملات خرابکارانه پی برد.

در تمامی مثال‌های بالا، نوع خاصی از داده‌ها به نام «داده‌های جاری» مطرح هستند که در قالب یک یا چند جریان‌داده منتقل می‌شوند.

۲.۲ چالش‌های پردازش و کاوش داده‌های جاری

داده‌های جاری در مقایسه با اشکال دیگر داده دارای خصوصیات منحصر به فردی هستند که پردازش و کاوش آن‌ها را به امری چالش‌برانگیز تبدیل می‌کند. از جمله‌ی این خصوصیات و چالش‌ها می‌توان به موارد زیر اشاره کرد:

۱ - نیاز به الگوریتم‌های تک-عبوره: با زیاد شدن حجم داده‌های جاری، پردازش بهینه‌ی داده‌ها به وسیله‌ی الگوریتم‌های چند-عبوره[۱۴] دیگر امکان‌پذیر نخواهد بود. بنابراین، الگوریتم‌ها و بسترها باید به گونه‌ای طراحی شوند که با یک بار عبور از داده‌ها، به نتایج مطلوب دست پیدا کنند.

۲ - نیاز به پردازش و کاوش کم-تأخیر: بسیاری از کاربردها نیازمند آن هستند که پردازش داده‌های جاری مرتبط با آن‌ها، به صورت بهنگام یا کم-تأخیر انجام شود. برای مثال، پست‌های مرتبط با یک خبر فوری در شبکه‌های اجتماعی فقط در بازه‌ی زمانی کوتاهی باارزش هستند. همچنین، در صورت وقوع یک تصادف در یک بزرگراه، تصمیم‌گیری هرچه سریعتر به کاهش تبعات نامطلوب منجر خواهد شد. مواردی مانند نظارت پزشکی[۱۵] و تشخیص ناهنجاری و حملات در شبکه‌های کامپیوتری هم از این دست کاربردها هستند.

۳ - عدم امکان ذخیره‌ی همه‌ی داده‌ها بر روی حافظه‌های انبوه و پایگاه داده‌ها: با گذشت زمان، حجم داده‌ها ممکن است به قدری زیاد شود که عملا ذخیره‌سازی آن‌ها بر روی دیسک و حافظه‌های انبوه امکان‌پذیر نباشد. از طرف دیگر، به دلیل سربار زیاد دسترسی به حافظه‌های انبوه و دیسک‌ها، پردازش و کاوش کم-تأخیر داده‌ها نیازمند آن است که پردازش داده‌ها در حافظه‌ی اصلی صورت گیرد و نیاز به دسترسی به حافظه‌های انبوه به حداقل برسد. این نیازمندی همچنین سبب می‌شود که در بسیاری از موارد، سامانه‌های پردازشی به صورت توزیع‌یافته[۱۶] و مبتنی بر بسترهای رایانش ابری طراحی و پیاده‌سازی شوند.

۴ - امکان تغییر در نرخ ورود[۱۷] و حجم[۱۸] داده‌ها: سرعت و حجم ورود داده‌های یک جریان‌های داده هم ممکن است در طول زمان تغییر کند. برای مثال، نرخ ورود داده‌ها به یک سامانه‌ی کنترل ترافیک جاده‌های بین شهری، در روزهای عادی با تعطیلات بسیار متفاوت است.

---

[۱۴] Multi-pass Algorithms

[۱۵] Medical Monitoring

[۱۶] Distributed

[۱۷] Input Rate

[۱۸] Volume

۵ – وقوع تحول[۱۹] در داده‌ها : بسیاری از جریان‌های داده، در طول زمان دچار تحول می‌شوند. یک مثال خوب برای تحول، تغییر در توزیع[۲۰] کلاس‌های مختلف داده در جریان داده می‌باشد. بنابراین، الگوریتم‌های پردازش داده‌های جاری باید به گونه‌ای طراحی و پیاده‌سازی شوند که وقوع تحول در طول زمان باعث کاهش کارآیی آن‌ها نشود.

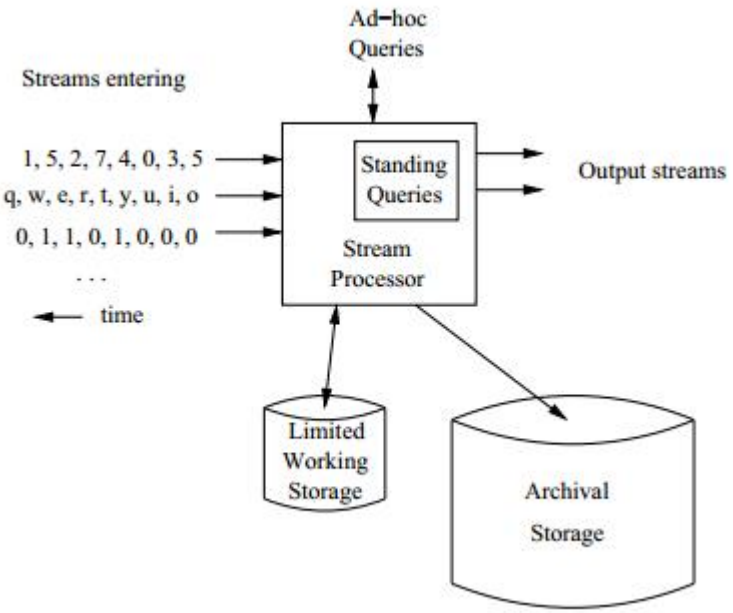
۲.۳ مدل کلاسیک پردازش داده‌های جاری

در شکل ۱ شمای کلی یک سامانه‌ی پردازش داده‌های جاری نشان داده شده است. مطابق شکل، جریانی از داده‌ها وارد سامانه شده و بخشی از آن‌ها که برای پردازش مورد نیاز است در یک پایگاه داده محدود نگهداری می‌شوند. سپس پردازش‌های لازم روی این داده‌ها انجام شده و نتایج حاصل از آن در قالب یک جریان‌داده خروجی تولید می‌شوند. پردازشگر جریان[۲۱] بخش اصلی این سامانه است. در این پروژه، الگوریتم‌های پردازش و کاوش بر بستر یک پردازشگر جریان پیاده‌سازی خواهند شد.

[۱۹] Evolution

[۲۰] Distribution

[۲۱] Stream Processor



شکل ۱- شمای کلی یک سامانه‌ی پردازش داده‌های جاری [۲]

با توجه به چالش‌های ذکر شده و نیازهای روزافزون به پردازش و کاوش داده‌های جاری، انتخاب بستری مناسب برای پیاده‌سازی الگوریتم‌های داده‌کاوی مورد نظر به تصمیمی مهم بدل می‌شود. در ادامه، مطرح‌ترین بسترهای موجود برای این کار معرفی و بررسی می‌شوند.

۲.۴ بسترهای توزیع‌یافته‌ی پردازش داده‌های جاری

با توجه به خصوصیات منحصربه‌فرد داده‌های جاری (که در بخش ۲,۲ مورد بررسی قرار گرفت) پردازش داده‌های جاری در بسیاری از کاربردهای موردنظر نیازمند بسترهایی توزیع‌یافته می‌باشد. در همین راستا، در سالیان اخیر بسترهای توزیع‌یافته‌ی مختلفی تولید شده است که سه مورد از مهم‌ترین آن‌ها عبارتند از آپاچی فلینک[۲۲]، آپاچی استورم[۲۳]، و آپاچی اسپارک. در این قسمت، ابتدا معماری لایه‌ای معمول بسترهای توزیع‌یافته پردازش داده‌های جاری مورد بررسی قرار خواهد گرفت. در ادامه، با توجه به معماری لایه‌ای معرفی شده مرور مختصری از هر یک از این سه بستر آورده می‌شود و پس از آن، بستر انتخابی

[۲۲] Apache Flink

[۲۳] Apache Storm

برای پیاده‌سازی الگوریتم‌های داده‌کاوی در این پروژه – آپاچی اسپارک - و دلیل این انتخاب بیان خواهد شد.

۲.۴.۱ معماری عمومی بسترهای توزیع‌یافته‌ی پردازش داده‌های جاری

شکل ۲ یک معماری لایه‌ای برای بسترهای توزیع‌یافته‌ی پردازش داده‌های جاری را نشان می‌دهد. بسیاری از بسترهای مطرح پردازش داده‌های جاری (مانند آپاچی فلینک، آپاچی استورم، و آپاچی اسپارک) به نوعی برپایه‌ی این معماری توسعه یافته‌اند [۱۴]. مدل مورد بحث از چهار لایه‌ی رابط برنامه‌نویسی گراف کاربر[۲۴]، طرح منطقی[۲۵]، طرح اجرایی[۲۶]، و ارتباطات شبکه[۲۷] تشکیل شده است. همچنین یک پیکرپاره[۲۸] برای مدیریت منابع گره‌های مختلف در این مدل به کار می‌رود. از جمله‌ی رایج‌ترین مدیرهای منابع[۲۹] می‌توان به مِسوز[۳۰] و یارن[۳۱] اشاره کرد. در ادامه، لایه‌های مورد بحث به صورت اجمالی بررسی می‌شوند.

[۲۴] User Graph API

[۲۵] Logical Plan

[۲۶] Executive Plan

[۲۷] Network Communications

[۲۸] Component

[۲۹] Resource Manager

[۳۰] Mesos

[۳۱] Yet Another Resource Negotiator (YARN)

شکل ۲- معماری لایه‌ای بسترهای توزیع‌یافته پردازش داده‌های جاری

بالاترین لایه، رابط برنامه‌نویسی گراف کاربر است. این لایه، رابط برنامه‌نویسی کاربردی‌ای[۳۲] برای برنامه‌های کاربردی پردازش و کاوش جریان‌داده‌ها فراهم می‌کند. کاربران با استفاده از این رابط برنامه‌نویسی می‌توانند برنامه‌های کاربردی خود را به صورت گراف‌هایی مدل کنند که رأس‌های آن‌ها، گره‌های پردازشی هستند و رویدادها از طریق یال‌ها بین گره‌ها جریان پیدا می‌کنند.

لایه‌ی دوم، طرح منطقی نام دارد و در واقع گراف تبدیل‌یافته‌ای از گراف تعریف شده توسط کاربر (لایه‌ی اول) می‌باشد. تبدیل فوق توسط موتور پردازشی بستر و با توجه به محیط اجرایی[۳۳] صورت می‌گیرد و سپس گراف حاصل در خوشه‌ای از گره‌های پردازشی - لایه‌ی سوم - توزیع می‌شود.

لایه‌ی چهارم به مدیریت ارتباطات و شبکه‌ی بین گره‌های پردازشی مختلف – که ممکن است در خوشه‌های مختلفی قرار گرفته باشند – می‌پردازد. این لایه همچنین وظیفه‌ی سریالیزه کردن[۳۴] اشیاء و انتقال آن‌ها در شبکه با استفاده از پروتکل‌هایی مانند TCP، و کنترل جریانِ[۳۵] داده‌ها را برعهده دارد.

---

[۳۲] Application Programming Interface (API)

[۳۳] Runtime Environment

[۳۴] Serialization

[۳۵] Flow Control

در نهایت، یک مدیر منابع وظیفه‌ی اداره‌ی منابع پردازشی مختلف، و زمان‌بندی وظایف[۳۶] میان خوشه‌ها و گره‌ها را برعهده دارد. بسیاری از بسترهای توزیع‌یافته‌ی پردازش داده‌های جاری برای این منظور از برنامه‌های مدیریت منابعی مانند مسوز، یارن، و نیمبِس[۳۷] استفاده می‌کنند.

موضوع مهم دیگری که در طراحی و استفاده از بسترهای توزیع‌یافته‌ی پردازش داده‌های جاری مورد توجه قرار می‌گیرد، تضمین‌های پردازش[۳۸] و نحوه‌ی ترمیم پس از وقوع خرابی[۳۹] می‌باشد. در پردازش‌های توزیع‌یافته در مقیاس بزرگ، خطاها ممکن است به دلایل مختلفی، مانند خرابی گره‌ها، خرابی شبکه، اشکالات نرم‌افزاری، و محدودیت منابع رخ دهند. از آنجا که یکی از نیازمندی‌های پردازش داده‌های جاری، پردازش بهنگام یا کم‌تأخیر است، در صورت وقوع خرابی و خطا، سامانه پردازشی باید بتواند به سرعت خطا را رفع کرده و پردازش را ادامه دهد. همچنین، وقوع خطا حتی‌الامکان نباید تأثیری در نتیجه‌ی پردازش داشته باشد. تضمین‌های پردازش، با توجه به نحوه‌ی ترمیم پس از وقوع خرابی در سامانه‌ی موردنظر تعریف می‌شوند.

به طور کلی، تضمین‌های پردازش در موتورهای پردازش جریان داده‌ها بر سه نوع هستند[۱۴]:

۱. دقیقا یک بار[۴۰]: این نوع از تضمین با ترمیم دقیق[۴۱] همراه است. پس از ترمیم دقیق، به جز افزایش مقطعی تأخیر، هیچ اثری از وقوع خرابی باقی نمی‌ماند و تمامی داده‌ها دقیقا یک بار پردازش می‌شوند.

[۳۶] Tasks

[۳۷] Nimbus

[۳۸] Processing Guarantees

[۳۹] Recovery from Failures

[۴۰] Exactly Once

[۴۱] Precise Recovery

۲. حداقل یک بار[۴۲]: این تضمین با ترمیم با عقبگرد[۴۳] متناظر است. در ترمیم با عقبگرد، هیچ بخشی از داده‌های جریان ورودی سامانه از بین نمی‌رود ولی وقوع خرابی ممکن است اثرات دیگری علاوه بر افزایش مقطعی تأخیر داشته باشد. در این صورت، ممکن است بعضی از داده‌ها دوباره (بیش از یک بار) پردازش شوند. به همین دلیل، این نوع تضمین، «حداقل یک بار» نام گرفته است.

بدون تضمین[۴۴]: در صورت استفاده از روش ترمیم شکافی[۴۵]، در صورت وقوع خرابی ممکن است بخشی از جریان ورودی به سامانه از بین برود. لذا در این حالت تضمینی برای پردازش همه‌ی داده‌ها وجود ندارد.

قسمت بعدی این بخش به معرفی و بررسی سه مورد از مطرح‌ترین بسترهای توزیع‌یافته‌ی پردازش داده‌های جاری – آپاچی فلینک، آپاچی استورم، و آپاچی اسپارک - اختصاص دارد.

۲.۴.۲ آپاچی فلینک

آپاچی فلینک بستری توزیعی برای پردازش دسته‌ای داده‌های عظیم و داده‌های جاری است [۵]. این پروژه در سال ۲۰۱۰ در آلمان و با بودجه‌ی بنیاد تحقیقات آلمان و با نام استراتوسفر[۴۶] آغاز به کار کرد و از سال ۲۰۱۴ به عنوان یک پروژه‌ی سطح‌بالای بنیاد آپاچی[۴۷] مطرح شده است. فلینک برای پردازش داده‌های جاری، رابط برنامه‌نویسی نرم‌افزاری به نام DataStream API دارد که با استفاده از آن می‌توان

[۴۲] At Leat Once

[۴۳] Rollback Recovery

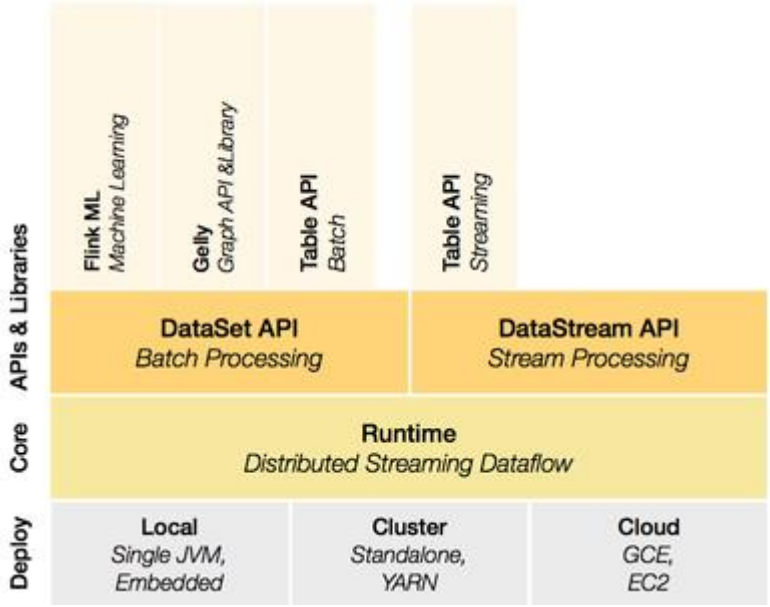
[۴۴] No Guarantee

[۴۵] Gap Recovery

[۴۶] Stratosphere

[۴۷] Apache Foundation Top-Level Project

به زبان‌های جاوا[۴۸] و اسکالا[۴۹] برنامه نوشت. شکل ۳، معماری آپاچی فلینک را نشان می‌دهد که شامل سه لایه‌ی استقرار[۵۰]، هسته[۵۱] و رابط‌های برنامه‌نویسی و کتابخانه‌ها می‌باشد.



شکل ۳ - معماری لایه‌ای آپاچی فلینک [۵]

فلینک تضمین پردازش دقیقا یک بار را فراهم کرده و برای ترمیم پس از وقوع خرابی از حالت‌برداری[۵۲] استفاده می‌کند.

[۴۸] Java

[۴۹] Scala

[۵۰] Deployment

[۵۱] Core

[۵۲] Checkpointing

۲.۴.۳ آپاچی استورم

آپاچی استورم یک بستر محاسباتی توزیع‌یافته و تحمل‌پذیرخطا[۵۳] برای محاسبات بهنگام در حجم وسیع است [۶] که بخش عمده‌ی آن به زبان برنامه‌نویسی کلوژر[۵۴] نوشته شده است. این پروژه ابتدا توسط تیمی در شرکت بک‌تایپ[۵۵] ایجاد شد. پس از مدتی شرکت توییتر این پروژه را خریداری کرده و آن را به صورت متن‌باز عرضه کرد. استورم از ماه سپتامبر سال ۲۰۱۴ به عنوان یک پروژه سطح بالای بنیاد آپاچی معرفی شده است.

کاربران استورم می‌توانند به صورت صریح گراف‌های کاربری مورد نیاز خود را با تعریف گره‌ها، نحوه‌ی توزیع و ارتباط بین آن‌ها مشخص کنند. این مورد یکی از تفاوت‌های اصلی استورم با فلینک و اسپارک است، که در آنها امکان تعریف صریح گراف‌های کاربری وجود ندارد و خود موتور اجرایی با توجه به تعاریف سطح بالاتر صورت گرفته توسط کاربران این کار را انجام می‌دهد. استورم فاقد مکانیزم کنترل جریان است که این امر می‌تواند به ازدحام[۵۶] در میان‌گیرهای[۵۷] ورودی یا از دست رفتن داده‌های ورودی منجر شود. در زمینه‌ی تضمین‌های پردازشی، استورم تضمین حداقل یک بار پردازش را فراهم می‌کند.

۲.۴.۴ آپاچی اسپارک

آپاچی اسپارک یک موتور سریع وعام‌منظوره برای پردازش داده‌ها در مقیاس بزرگ است [۷][۸][۹]. اسپارک به عنوان موتور پردازشگر در استک تحلیل داده‌های برکلی[۵۸] مطرح می‌شود [۱۰] و

---

[۵۳] Fault-tolerant

[۵۴] Clojure

[۵۵] BackType Inc.

[۵۶] Congestion

[۵۷] Buffers

[۵۸] BDAS the Berkeley Data Analytics Stack

می‌توان برای آن به زبان‌های جاوا، اسکالا، پایتون[۵۹]، و آر[۶۰] برنامه نوشت. اسپارک شامل تعدادی رابط برنامه‌نویسی نرم‌افزار برای پردازش داده‌های جاری (اسپارک‌استریمینگ[۶۱]) [۱۱]، کار با داده‌های ساختارمند[۶۲] (اسپارک سیکوئل[۶۳])، یادگیری ماشین[۶۴] (ام‌ال‌لیب[۶۵] [۱۸])، و پردازش گراف[۶۶] (گراف‌اکس[۶۷]) می‌باشد. در شکل ۴ محل قرارگیری اسپارک و رابط‌های برنامه‌نویسی آن در استک تحلیل داده‌های برکلی نشان داده شده است. اجزای آبی‌رنگ در ابتدا در آزمایشگاه امپ‌لب[۶۸] دانشگاه برکلی[۶۹] توسعه داده شده‌اند و اسپارک هم یکی از همین موارد است.

[۵۹] Python

[۶۰] R

[۶۱] Spark Streaming

[۶۲] Structured Data

[۶۳] Spark SQL

[۶۴] Machine Learning

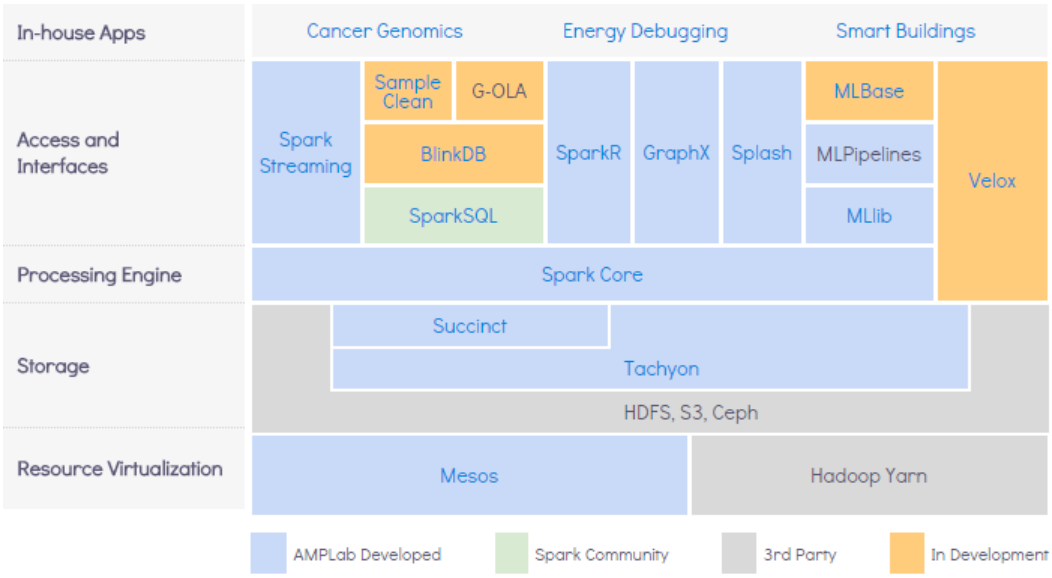
[۶۵] MLlib

[۶۶] Graph Processing

[۶۷] GraphX

[۶۸] AMPLab

[۶۹] University of California, Berkeley



شکل ۴ – استک تحلیل داده‌های برکلی [۱۰]

داده‌ساختار[۷۰] اصلی اسپارک برای کار با داده‌های حجیم و داده‌های جاری، مجموعه‌داده‌ی ارتجاعی توزیع‌یافته[۷۱] (RDD) نام دارد [۸]. RDD ها مجموعه‌هایی تغییرناپذیر[۷۲] از اشیا و تحمل‌پذیر خطا هستند که بر روی یک خوشه[۷۳] توزیع شده‌اند. می‌توان گفت رابط برنامه‌نویسی کاربردی اسپارک بر مبنای تعریف RDDها و استفاده از عملگرهای مخصوص آن‌ها توسعه داده شده است.

همانند فلینک و برخلاف استورم، کاربران اسپارک نمی‌توانند گراف کاربری را به صورت صریح تعریف کنند. در عوض، موتور اجرایی اسپارک با توجه به عملگرهای استفاده شده در برنامه‌ی کاربردی[۷۴] گراف موردنظر را ایجاد می‌کند.

---

۷۰ Data Structure

۷۱ Resilient Distributed Dataset

۷۲ Immutable

۷۳ Cluster

۷۴ Application

در حقیقت، اسپارک یک موتور پردازش دسته‌ای[۷۵] داده‌ها است و در نتیجه برای پردازش داده‌های جاری، آن‌ها را به دسته‌های کوچکی تقسیم کرده و سپس اعمال پردازشی لازم را روی هر دسته انجام می‌دهد. به طور مشخص در مورد پردازش داده‌های جاری، یک جریان داده‌ی ورودی با داده‌ساختار دیگری به نام جریانِ گسسته‌شده[۷۶] (DStream) متناظر می‌شود که در واقع دنباله‌ای از RDD هاست.

RDDها از دو دسته اعمال پشتیبانی می‌کنند. دسته‌ی اول، تبدیل‌ها[۷۷] هستند که از یک RDD موجود، یک RDD جدید ایجاد می‌کنند. دسته‌ی دیگر اعمال، اقدام‌ها[۷۸] هستند که پس از انجام پردازش روی داده‌ها، یک مقدار را به عنوان خروجی به برنامه‌ی گرداننده[۷۹] بر می‌گردانند. برای مثال، یکی از معمول‌ترین تبدیل‌ها، تبدیل map می‌باشد که یک تابع را به تمامی اعضای یک مجموعه‌داده اعمال می‌کند و مجموعه‌داده‌ی جدیدی – که هر عضو آن، نتیجه‌ی اعمال تابع موردنظر بر اعضای مجموعه‌داده‌ی ورودی است - تولید می‌کند. count هم مثالی از یک اقدام است، که تعداد اعضای یک مجموعه‌داده را محاسبه کرده و به عنوان خروجی برمی‌گرداند.

لازم به ذکر است که تبدیل‌های موجود در اسپارک، اصطلاحا تنبل[۸۰] هستند، یعنی این تبدیل‌ها تا زمانی که برروی مجموعه‌داده نهایی اقدامی صورت نگیرد انجام نمی‌شوند. در عوض، زنجیره‌ی تبدیل‌هایی که برروی یک مجموعه‌داده اعمال می‌شوند در قالب دودمان[۸۱] مجموعه‌داده‌ی نهایی نگهداری می‌شود و زمانی که در برنامه قرار شود یک اقدام روی مجموعه‌داده‌ی نهایی صورت بگیرد، تبدیل‌های موردنظر واقعا انجام می‌شوند تا مجموعه‌داده‌ی نهایی در عمل ایجاد شده و اقدام موردنظر بتواند روی آن صورت گیرد.

---

[۷۵] Batch Processing

[۷۶] Discretized Stream

[۷۷] Transformations

[۷۸] Actions

[۷۹] Driver Program

[۸۰] Lazy

[۸۱] Lineage

این کار به افزایش سرعت و بهینگی پردازش‌ها در اسپارک منجر می‌شود. همچنین، تحمل‌پذیری خطا و ترمیم پس از وقوع خرابی در اسپارک با استفاده از همین دودمان‌های نگهداری شده امکان‌پذیر می‌شود.

نوشتن برنامه‌های کاربردی و پیاده‌سازی الگوریتم‌های مختلف بر بستر آپاچی اسپارک نیازمند شناخت کامل مجموعه‌ی تبدیل‌ها و اقدام‌ها می‌باشد. در واقع، یک برنامه‌ی مبتنی بر اسپارک شامل زنجیره‌ای از تبدیل‌ها و اقدام‌های روی مجموعه‌داده‌هاست.

۲.۴.۵ انتخاب بستر مناسب برای پیاده‌سازی الگوریتم

در بخش‌های قبل، سه بستر آپاچی فلینک، آپاچی استورم، و آپاچی اسپارک مورد بررسی اجمالی قرار گرفته و از بعضی جنبه‌های فنی با یکدیگر مقایسه شدند. یکی از تصمیم‌های مهم برای تولید هر ابزاری، انتخاب بسترها و تکنولوژی‌های مورد استفاده برای پیاده‌سازی می‌باشد. در این بخش به دلیل انتخاب آپاچی اسپارک به عنوان بستر توزیع‌یافته پردازش داده‌های جاری و هسته‌ی اصلی ابزار SDMiner پرداخته می‌شود.

هرسه بستر ذکر شده به واسطه‌ی رابط‌های برنامه‌نویسی خود این امکان را به توسعه‌دهندگان می‌دهند که الگوریتم‌های مختلفی، از جمله الگوریتم‌های کاوش داده‌های جاری را بر روی آن‌ها پیاده‌سازی کنند. با توجه به اینکه هرسه بستر نسبتا جدید هستند، یک جنبه‌ی مهم در تصمیم‌گیری برای انتخاب بستر پیاده‌سازی، در دسترس بودن مستندات پیاده‌سازی و توسعه‌ی نرم‌افزار، و فعال‌بودن جامعه‌ی توسعه‌دهندگان می‌باشد. هرسه‌ی این بسترها به صورت متن‌باز و در گیت‌هاب[۸۲] موجود هستند. شکل ۵ تعداد تغییرات اعمال شده در کد[۸۳] در هر هفته را در بازه‌ی هفته‌ی اول فوریه‌ی ۲۰۱۵ تا پایان ژانویه ۲۰۱۶ (زمان شروع پیاده‌سازی پروژه) برای هر سه بستر نشان می‌دهد.

شکل ۵ – تعداد تغییرات اعمال شده در کد در هر هفته برای هر بستر در بازه‌ی فوریه‌ی ۲۰۱۵ تا ژانویه ۲۰۱۶

[۸۲] http://github.com

[۸۳] Code Commits

همان‌طور که مشاهده می‌شود، تعداد تغییرات اعمال شده در این بازه برای پروژه‌ی اسپارک به مراتب از استورم و فلینک بیشتر است. جدول ۱ به مقایسه‌ی برخی ویژگی‌های دیگر این سه بستر که در روند توسعه‌ی نرم‌افزارهایشان نقش مهمی دارند می‌پردازد. این اطلاعات از صفحات پروژه‌ها در وب‌سایت بنیاد آپاچی و همچنین مخازن کد گیت‌هاب متناظرشان و آمارهای وب‌سایت stackoverflow.com استخراج شده‌اند.

جدول۱ - مقایسه‌ی برخی ویژگی‌های مربوط به توسعه‌ی سه بستر (در تاریخ ۳۱ ژانویه ۲۰۱۶)

	آپاچی اسپارک	آپاچی استورم	آپاچی فلینک
توسعه‌دهندگان فعال	۷۹۷	۲۰۰	۱۵۴
سئوالات تگ‌شده در وب‌سایت stackoverflow.com	۹۹۰۰	۱۳۹۳	۲۰۸

با توجه به شکل ۵ و جدول ۱ و نظر به فعال‌تر بودن جامعه‌ی توسعه‌دهندگان و دردسترس‌تر بودن مستندات و منابع آموزشی، آپاچی اسپارک به عنوان بستر پیاده‌سازی الگوریتم در این پروژه انتخاب می‌شود.

در بخش بعدی، مرور مختصری از رابط برنامه‌نویسی کاربردی آپاچی اسپارک برای کار با داده‌های جاری (اسپارک‌استریمینگ) آورده شده است.

۲.۵ مروری بر رابط برنامه‌نویسی کاربردی اسپارک‌استریمینگ

اسپارک‌استریمینگ، رابط برنامه‌نویسی کاربردی آپاچی اسپارک برای پردازش داده‌های جاری است [۱۱]. شکل ۶ جریان کلی ورودی و خروجی داده‌ها در اسپارک‌استریمینگ را نشان می‌دهد. داده‌ها

می‌توانند از منابع مختلفی مانند آپاچی کافکا[۸۴] [۱۲]، توییتر[۸۵]، اچ‌دی‌اف‌اس[۸۶]، فلوم[۸۷]، و سوکت‌های TCP وارد شوند و پس از پردازش، خروجی را می‌توان برروی فایل‌سیستم‌های مختلف (مانند HDFS) و پایگاه‌های داده ذخیره کرد یا برروی داشبوردهای مختلف نمایش داد. در این پروژه، جریان‌داده‌های ورودی با استفاده از سوکت‌های TCP (به عنوان کلی‌ترین درگاه ورودی) خوانده می‌شوند و نتایج حاصل از وظایف داده‌کاوی بر روی یک داشبورد تحت وب نمایش داده می‌شود.

شکل ۶ - جریان کلی ورودی و خروجی در اسپارک‌استریمینگ [۱۱]

همانطور که در بخش ۲,۴,۴ بیان شد، اسپارک‌استریمینگ برای پردازش جریان‌داده‌ها، آن‌ها را به دسته‌های کوچکی تقسیم کرده و سپس اعمال پردازشی لازم را روی هر دسته انجام می‌دهد. شکل ۷ این موضوع را نشان می‌دهد.

---

[۸۴] Apache Kafka

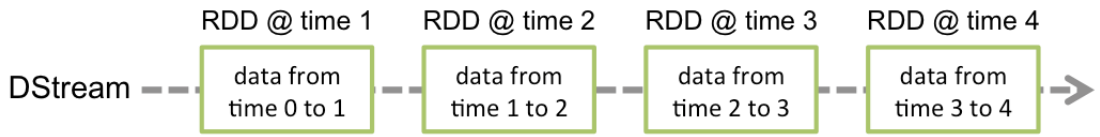
[۸۵] Twitter

[۸۶] HDFS

[۸۷] Flume

شکل ۷ - تقسیم جریان‌داده‌ی ورودی به دسته‌های داده در اسپارک‌استریمینگ

در اسپارک‌استریمینگ، برای پردازش داده‌های جاری، یک جریان داده‌ی ورودی با داده‌ساختار دیگری به نام جریان گسسته‌شده (DStream) متناظر می‌شود که در واقع دنباله‌ای از RDDهاست [۷][۱۶][۱۷]. هر RDD موجود در یک DStream شامل داده‌های ورودی در یک بازه‌ی زمانی[۸۸] مشخص است. RDDهای موجود در DStream بر اساس بازه‌ی زمانی متناظرشان مرتب شده‌اند. شکل ۸ این مورد را بهتر نشان می‌دهد.



شکل ۸- جریان گسسته‌شده و RDDهای موجود در آن

رابط برنامه‌نویسی اسپارک‌استریمینگ (و به طور کلی اسپارک) امکان نوشتن برنامه به زبان‌های اسکالا، جاوا، آر، و پایتون را برای برنامه‌نویسان و توسعه‌دهندگان فراهم می‌کند. با توجه به اینکه اسکالا یک زبان برنامه‌نویسی تابعی[۸۹] می‌باشد، نحو[۹۰] آن به خوبی با چارچوب تفکر تابعی حاکم بر اسپارک (اِعمال زنجیره‌ای از تبدیل‌ها و اقدام‌ها روی مجموعه‌داده‌ها) هماهنگ است و به همین دلیل به عنوان زبان برنامه‌نویسی برای پیاده‌سازی الگوریتم‌های کاوش داده‌های جاری در این پروژه انتخاب شده است. لازم به

[۸۸] Interval

[۸۹] Functional Programming Language

[۹۰] Syntax

ذکر است که اسکالا به طور کلی هم زبان اصلی مورد استفاده برای نوشتن برنامه‌های کاربردی مبتنی بر اسپارک بوده و پیاده‌سازی خود بستر آپاچی اسپارک نیز با استفاده از همین زبان صورت گرفته است.

۲.۶ خلاصه‌ی فصل

در این فصل مفاهیم پایه‌ی حوزه‌ی پردازش داده‌های جاری، چالش‌های این امر، و روش‌ها و معماری معمول بسترهای توزیع‌یافته برای پردازش و کاوش داده‌های جاری مورد بررسی قرار گرفت. سپس، سه بستر مطرح پردازش داده‌های جاری (آپاچی فلینک، آپاچی استورم، و آپاچی اسپارک) با توجه به مدل معرفی شده مورد بررسی قرار گرفتند و چگونگی انتخاب آپاچی اسپارک به عنوان بستر مورد استفاده در این پروژه شرح داده شد. در نهایت، به معرفی مختصری از رابط برنامه‌نویسی کاربردی آپاچی اسپارک برای کار با داده‌های جاری (اسپارک استریمینگ) پرداخته شد.

در فصل بعدی، الگوریتم نمونه‌برداری تصادفی بدون تبعیض، به عنوان یکی از معمول‌ترین الگوریتم‌های کاوش داده‌های جاری معرفی خواهد شد و چگونگی تبدیل آن به الگوریتمی توزیع‌یافته با توجه به چارچوب برنامه‌نویسی اسپارک‌استریمینگ مورد بررسی قرار خواهد گرفت.

فصل سوم –

الگوریتم نمونه‌برداری تصادفی توزیع‌یافته با مخزن ثابت

در این فصل به الگوریتم نمونه‌برداری تصادفی با مخزن ثابت[۹۱] (RSFR) به عنوان یکی از معمول‌ترین الگوریتم‌های کاوش داده‌های جاری پرداخته خواهد شد. در ابتدا مبحث نمونه‌برداری[۹۲] و کاربردهای آن در وظایف کاوش داده‌های جاری مورد بررسی قرار خواهد گرفت. پس از آن، به چند روش معمول برای نمونه‌برداری اشاره شده و الگوریتم RSFR شرح داده خواهد شد. سپس، در مورد فرآیند موازی‌سازی این الگوریتم و چگونگی طراحی و پیاده‌سازی نسخه‌ی توزیع‌یافته[۹۳] (DRSFR) بر بستر آپاچی اسپارک بحث خواهد شد.

۳.۱ نمونه‌برداری

نمونه‌برداری، یکی از روش‌های خلاصه‌سازی[۹۴] داده‌ها است [۱]. مسأله‌ی نمونه‌برداری عبارت است از انتخاب زیرمجموعه‌ای از داده‌ها به گونه‌ای که پاسخ‌های حاصل از پرس‌وجوهای[۹۵] صورت‌گرفته بر روی نمونه‌ی انتخاب شده به پاسخ‌های حاصل از پرس‌وجوهای صورت‌گرفته روی کل مجموعه داده‌ها نزدیک باشد. در صورتی که پرس‌وجوهای موردنیاز از قبل مشخص باشند می‌توان نمونه‌ها را با توجه به آن‌ها انتخاب کرد، اما در بسیاری از کاربردهای داده‌کاوی، پرس‌وجوهای تک‌کاره[۹۶] مطرح می‌شوند و به همین دلیل نمونه‌ی انتخاب شده باید دربرگیرنده‌ی تصویری کلی از مجموعه داده‌ها باشد [۲]. در مورد کاوش داده‌های جاری، با توجه به اینکه داده‌ها ممکن است در طول زمان دچار تحول شوند و حجم داده‌ها به نحوی است که نمی‌توان همه‌ی آن‌ها را در حافظه نگهداری کرد، انتخاب نمونه‌ی مناسب اهمیت بیشتری پیدا می‌کند.

---

[۹۱] Random Sampling with a Fixed Reservoir

[۹۲] Sampling

[۹۳] Distributed Random Sampling with a Fixed Reservoir

[۹۴] Synopsis Construction

[۹۵] Query

[۹۶] Ad-hoc

ویژگی اصلی نمونه‌برداری در مقایسه با سایر روش‌های خلاصه‌سازی داده‌ها – مانند تشکیل هیستوگرام[۹۷] یا موجک‌ها[۹۸] - سادگی و به‌صرفه‌بودن آن است. با استفاده از روش‌های نمونه‌برداری می‌توان به سادگی به تصویری بدون تبعیض[۹۹] از کل مجموعه داده‌ها با تضمین خطای قابل اثبات[۱۰۰] دست پیدا کرد. همچنین، روش‌های دیگر خلاصه‌سازی برای داده‌های چندبعدی[۱۰۱] به راحتی قابل استفاده نیستند و درواقع در کاربردهایی که با داده‌های چندبعدی سر و کار دارند، پراستفاده‌ترین روش خلاصه‌سازی، نمونه‌برداری – و به طور مشخص نمونه‌برداری تصادفی – می‌باشد [۱].

دو نمونه از پراستفاده‌ترین روش‌های نمونه‌برداری، نمونه‌برداری تصادفی با مخزن ثابت، و نمونه‌برداری مختصر[۱۰۲] می‌باشد. در این پروژه، الگوریتم نمونه‌برداری تصادفی با مخزن ثابت به عنوان اولین الگوریتم کتابخانه‌ی کاوش داده‌های جاری انتخاب، موازی‌سازی و پیاده‌سازی شده است. بخش بعدی به شرح این الگوریتم اختصاص دارد.

۳.۲ الگوریتم نمونه‌برداری تصادفی با مخزن ثابت (RSFR)

در این بخش، الگوریتم RSFR مورد بررسی قرار خواهد گرفت. هدف از اجرای این الگوریتم، به دست‌آوردن نمونه‌ای بدون تبعیض با اندازه‌ی مشخص (مخزن ثابت) از جریان‌داده‌ی ورودی می‌باشد. شکل ۹ گام‌های اجرای این الگوریتم را نشان می‌دهد.

با فرض مخزن با سایز n، در ابتدا n عضو اول جریان‌داده در مخزن قرار داده می‌شوند. وقتی عضو k ام جریان‌داده وارد می‌شود، با احتمال n/k برای قرارگیری در مخزن انتخاب می‌شود. در صورت انتخاب

---

[۹۷] Histogram Construction

[۹۸] Wavelets

[۹۹] Unbiased

[۱۰۰] Provable Error Guarantees

[۱۰۱] Multi-dimensional Data

[۱۰۲] Concise Sampling

شدن عضو k ام، چون سایز مخزن ثابت درنظر گرفته شده، برای قرارگیری در مخزن باید با یک عضو موجود در مخزن جایگزین شود. عضوی که باید از مخزن خارج شود با احتمال 1/n از میان تمام اعضای فعلی مخزن انتخاب خواهد شد و سپس عضو k ام ورودی درجای آن قرار خواهد گرفت.

Random Sampling with a Fixed Reservoir (RSFR)

We have a reservoir of size **n**.
Input data comes in form of a stream of elements.

---

1 Add the first ***n*** elements of the data stream to the reservoir for initialization.

2 When the ***k***th element arrives, it is placed in the reservoir with a probability of ***n*/*k***.

3 If ***k***th element has to be added to the reservoir, an existing element of the reservoir with equal probability of 1/***n*** will be selected and removed from the stream, and the ***k***th element of input will replace it.

شکل ۹- گام‌های اجرای الگوریتم نمونه‌برداری تصادفی با مخزن ثابت

با استقرا برروی k می‌توان به این نتیجه رسید که خروجی الگوریتم RSFR یک نمونه‌ی بدون تبعیض از جریان‌داده است. اگر جریان‌داده برای مدت کافی ادامه پیدا کند، k خیلی بزرگ شده و به سمت بی‌نهایت میل می‌کند، پس تمامی اعضای جریان‌داده با احتمال یکسان می‌توانند در مخزن (نمونه) قرار بگیرند.

با وجود اینکه با استفاده از الگوریتم فوق می‌توان در هر زمان به نمونه‌ای بدون تبعیض از جریان‌داده‌ی ورودی دست پیدا کرد، در طراحی آن فقط یک گره درنظر گرفته شده که تمامی داده‌ها وارد آن می‌شوند، محاسبات در آن گره انجام می‌شود و نمونه را با توجه به محاسبات انجام شده تغییر می‌دهد. لذا نمی‌توان آن را برروی سیستم‌های توزیع‌یافته استفاده کرد و این با نیازمندی‌های کاوش و پردازش داده‌های جاری در تضاد است. از طرف دیگر، الگوریتم RSFR حساس به ورود عضو جدید جریان‌داده است، یعنی محاسبه احتمالات و به‌روزرسانی احتمالی مخزن با ورود هر عضو جدید به گره صورت می‌گیرد. این در حالی است که رابط برنامه‌نویسی اسپارک‌استریمینگ داده‌های ورودی را در بازه‌های زمانی ثابتی

جمع‌آوری کرده و در قالب RDD قرار می‌دهد و انجام تبدیلات و اقدام‌ها فقط روی این RDDها امکان‌پذیر است و نمی‌توان در هر لحظه‌ای که یک عضو جدید وارد شد عملیات موردنظر را انجام داد.

با توجه به توضیحات فوق، الگوریتم RSFR باید به گونه‌ای بازنویسی شود که هم در مدل برنامه‌نویسی اسپارک‌استریمینگ قابل پیاده‌سازی باشد و هم از رایانش توزیع‌یافته پشتیبانی کند. در این پروژه، الگوریتم RSFR برای برآورده کردن نیازمندی‌های فوق، بازطراحی شده و الگوریتم حاصل، «الگوریتم نمونه‌برداری تصادفی توزیع‌یافته با مخزن ثابت (DRSFR)» نام گرفته است.

در بخش بعدی الگوریتم DRSFR مورد بررسی قرار خواهد گرفت.

۳.۳ الگوریتم نمونه‌برداری تصادفی توزیع‌یافته با مخزن ثابت (DRSFR)

الگوریتم DRSFR در واقع توسعه‌ای از الگوریتم RSFR است که دو خصوصیت زیر را به آن اضافه می‌کند:

- موازی‌شده است و از رایانش توزیع‌یافته پیشتیبانی می‌کند، و
- با توجه به مدل برنامه‌نویسی اسپارک‌استریمینگ که داده‌های ورودی را به صورت دسته‌ای در بازه‌های زمانی مشخص در اختیار گرداننده قرار می‌دهد، قابل پیاده‌سازی است.

در این بازطراحی منطق عملکرد الگوریتم RSFR تغییری نمی‌کند و در نتیجه اثبات بدون تبعیض بودن این الگوریتم که در بخش ۳,۲ ذکر شد به قوت خود باقی است.

همان‌طور که در بخش قبلی توضیح داده شد، الگوریتم RSFR، احتمالات مورد نظر را محاسبه کرده و در صورت لزوم اقدام به به‌روزرسانی مخزن (نمونه) می‌کند. اسپارک‌استریمینگ داده‌های ورودی را در قالب RDD هایی که مربوط به بازه‌های زمانی مشخص هستند در اختیار برنامه‌ی کاربردی قرار می‌دهد (شکل ۸). ترتیب قرارگیری داده‌ها در RDDها همان ترتیب ورودشان به سامانه است ولی داده‌ها بر این اساس اندیس‌گذاری[۱۰۳] نمی‌شوند.

[۱۰۳] Indexing

بسیاری از منابع جریان‌داده (برای مثال سنسورهای سنجش کیفیت هوا) خود اقدام به زدن برچسب زمانی[۱۰۴] یا شماره‌گذاری داده‌های ورودی (بر اساس زمان و ترتیب تولید داده‌ها) می‌کنند، ولی خروجی دسته‌ی دیگر تولیدکننده‌های جریان‌داده فقط داده‌ی تولید شده (بدون برچسب و شماره) است. یک راه‌حل جامع باید بتواند از هر دو دسته‌ی داده‌های ورودی پشتیبانی کند، پس باید بتوان داده‌های ورودی را به ترتیب ورودشان شماره‌گذاری کرد تا محاسبه‌ی احتمالات مربوط به الگوریتم RSFR امکان‌پذیر شود، چون موتور پردازشی اسپارک بسته به تعداد گره‌های اجرایی، داده‌های ورودی را بین آن‌ها پخش می‌کند و ممکن است در بازگشت به برنامه‌ی گرداننده، ترتیب داده‌ها به هم بخورد.

در این پروژه، دو پیاده‌سازی برای الگوریتم DRSFR ارائه شده است. پیاده‌سازی اول مربوط به حالتی است که داده‌های ورودی از مبدا به ترتیب تولید شدنشان (و در نتیجه ترتیب ورودشان) شماره‌گذاری شده‌اند و به شماره‌گذاری مجدد نیازی نیست. پیاده‌سازی دوم، حالت کلی را که در آن داده‌ها بدون شماره‌گذاری هستند را پوشش می‌دهد و با استفاده از روش MapWithState اسپارک‌استریمینگ، داده‌ها به ترتیب ورودشان به سیستم اندیس‌گذاری می‌شوند. در ادامه به جزئیات هر دو پیاده‌سازی پرداخته می‌شود.

۳.۳.۱ پیاده‌سازی DRSFR برای داده‌های جاری شماره‌گذاری شده

پیش‌فرض این پیاده‌سازی این است که داده‌های ورودی به صورت رشته‌ای متشکل از مقدار اصلی و اندیس هستند. شکل ۱۰ گام‌های اجرای این الگوریتم را نمایش می‌دهد.

در انتهای هر بازه‌ی زمانی، DStream ورودی با RDD های حاوی داده‌های ورودی در آن بازه به‌روزرسانی می‌گردد. کار با تبدیل رشته‌ی ورودی متناظر با هر عضو DStream به یک زوج مرتب (value, index) آغاز می‌شود. به عبارت دیگر، index در نقش k مورد بحث در الگوریتم RSFR می‌باشد و حال می‌توان از آن برای محاسبه‌ی احتمالات موردنیاز استفاده کرد.

[۱۰۴] Timestamping

در مرحله‌ی بعد، یک عمل صافی[۱۰۵] بر روی زوج‌مرتب‌ها (که به ترتیب ورودشان در RDD قرار گرفته‌اند) انجام می‌شود تا فقط زوج‌مرتب‌هایی باقی بمانند که شرط احتمالاتی موردنظر (قرارگیری در مخزن با احتمال n/index) را برآورده می‌کنند (لازم به ذکر است که برای n عضو اول، شرط احتمالی موردنظر همواره برقرار است، پس نیازی به قدم جداگانه‌ای برای درج اعضای اولیه نمونه نیست). با توجه به اینکه تعداد زوج‌مرتب‌های باقی‌مانده بسیار کمتر از کل مجموعه داده‌هاست، می‌توان عملیات درج آن‌ها در مخزن (نمونه) را برروی گره برنامه‌ی گرداننده انجام داد. بدین منظور، از یک عمل جمع‌آوری[۱۰۶] استفاده می‌شود تا دسته‌های داده‌ی باقی‌مانده‌ی پخش شده در گره‌های مختلف، در یک مجموعه در گره اصلی (برنامه‌ی گرداننده) گردآوری شوند. سپس عملیات مرتب‌سازی صعودی این مجموعه بر اساس اندیس اعضا صورت می‌گیرد، و در نهایت با شروع از اول مجموعه‌ی مرتب شده، به ازای هر عضو این مجموعه، یک عضو قدیمی موجود در مخزن از آن خارج شده و عضو جدید موردنظر در جای آن قرار می‌گیرد.

[۱۰۵] Filter

[۱۰۶] Collect

Distributed Random Sampling with a Fixed Reservoir (DRSFR)

For Pre-Indexed Data Streams and Apache SparkStreaming

We have a reservoir of size **n**.

random is a floating point number between 0 and 1, randomly generated on each occurrence.
Input data comes in form of a stream of pre-indexed elements. Indices are based on the order of production of data elements.
Input elements are in the form of: **"value, index"**

---

foreach batch interval ***interval***, do the following:

1 **transform** the ***interval.DStream*** into another DStream by applying a **map** operation to the containing RDD, resulting in a new DStream containing a RDD with its elements in the form of (value, index), and name the new DStream as ***indexedDStream***.

2 **filter** ***indexedDStream*** **foreach** element ***e*** in ***indexedDStream*** that satisfies the following predicate, and name the filtered DStream as ***filteredDStream***:

(n / ***e***.index) > ***random***

3 **foreach** RDD ***r*** in ***filteredDStream***, do the following:

3.1 **collect** ***r*** inside an array named ***updateSet***.

3.2 **sort** ***updateSet*** in ascending order based on indices of elements of ***r***, call the new array as ***sortedUpdateSet.***

3.3 **foreach** element ***el*** in ***sortedUpdateSet***, do the following:

3.3.1 ***replaceIndex*** = a random integer between 0 and n-1.

3.3.2 replace the reservoir element with the index of ***replaceIndex*** with ***el.***

شکل ۱۰- گام‌های اجرای الگوریتم DRSFR برای داده‌های شماره‌گذاری شده

۳.۳.۲ پیاده‌سازی DRSFR برای داده‌های جاری بدون شماره

تفاوت این حالت با حالت قبل در این است که داده‌های ورودی به صورت رشته‌هایی فقط حاوی مقدار اصلی (و بدون شماره) هستند، و با توجه به پخش شدن داده‌ها بر روی گره‌های مختلف (پس از

ورود)، باید نسبت به شماره‌گذاری ترتیبی آن‌ها اقدام نمود. گام‌های اجرای این نسخه از الگوریتم با حالت قبلی فقط در گام شماره‌ی یک شکل ۱۰ تفاوت دارد. در این حالت، با استفاده از مفهوم وضعیت[۱۰۷] در اسپارک‌استریمینگ، هر عضو با شماره‌ی ورودش متناظر شده و اندیس‌گذاری می‌شود. باقی مراحل الگوریتم مانند شکل ۱۰ خواهد بود. این نسخه از الگوریتم، کلی‌ترین حالت موجود است و برای تمامی داده‌هایی که می‌توانند به صورت رشته‌ای وارد سامانه شوند (کاراکترها، مقادیر عددی و ...) قابل استفاده است.

۳.۴ خلاصه‌ی فصل

در این فصل ابتدا به کاربردهای نمونه‌برداری در وظایف کاوش و پردازش داده‌های جاری پرداخته شد. سپس به تعدادی از الگوریتم‌ها و روش‌های نمونه‌برداری و خلاصه‌سازی جریان‌داده‌ها اشاره شده و الگوریتم نمونه‌برداری تصادفی با مخزن ثابت (RSFR) به عنوان یکی از معمول‌ترین الگوریتم‌های نمونه‌برداری جریان‌داده‌ها مورد بررسی قرار گرفت. در نهایت نیز به طراحی و پیاده‌سازی نسخه‌ی توزیع‌یافته‌ی الگوریتم (DRSFR) با توجه به مدل برنامه‌نویسی اسپارک‌استریمینگ پرداخته شد. الگوریتم DRSFR اولین الگوریتم پیاده‌سازی شده در کتابخانه‌ی الگوریتم‌های داده‌کاوی این پروژه است.

فصل بعدی به طراحی و پیاده‌سازی ابزار مبتنی بر آپاچی اسپارک برای کاوش داده‌های جاری (SDMiner)، متدولوژی مهندسی نرم‌افزار به کار رفته در این پروژه، و نتایج حاصل از پیاده‌سازی اختصاص خواهد داشت.

[۱۰۷] State

فصل چهارم –
طراحی، پیاده‌سازی و ارزیابی

در این فصل به طراحی، پیاده‌سازی و تحلیل مهندسی نرم‌افزار ابزار SDMiner پرداخته خواهد شد. بدین منظور، در ابتدا معماری کلی ابزار بیان شده و سپس به بحث در مورد هر یک از اجزای این ابزار پرداخته می‌شود. در خلال بررسی اجزای مختلف، به فناوری‌های مورد استفاده در پیاده‌سازی آن‌ها اشاره می‌شود. پس از آن، مستندات طراحی و متدولوژی مهندسی نرم‌افزار به کار رفته برای انجام این پروژه مورد بررسی قرار خواهد گرفت.

۴.۱ معماری و پیکرپاره‌های ابزار SDM ner

هدف از طراحی و پیاده‌سازی ابزار SDMiner، ساده‌تر کردن تعریف و اجرای وظایف کاوش داده‌های جاری بوده است. به همین منظور، معماری لایه‌ای شکل ۱۱ برای این ابزار درنظر گرفته و پیکرپاره‌های مختلف با توجه به این معماری طراحی و پیاده‌سازی شدند.

شکل ۱۱ – معماری لایه‌ای SDM ner

در بالاترین لایه، رابط کاربری ابزار قرار دارد که می‌توان با استفاده از مرورگرهای وب به آن دسترسی داشت. پیاده‌سازی این رابط کاربری با استفاده از اچ‌تی‌ام‌ال[۱۰۸] و سی‌اس‌اس[۱۰۹] صورت گرفته است.

در لایه‌ی بعدی، کتابخانه‌ی الگوریتم‌های کاوش داده‌های جاری جای گرفته است که شامل فایل‌های پیاده‌سازی الگوریتم‌های مختلف – با توجه به مدل برنامه‌نویسی اسپارک‌استریمینگ – می‌باشد. کاربر می‌تواند الگوریتم موردنظر خود را انتخاب کرده، پارامترهای موردنیاز الگوریتم و برنامه‌ی کاربردی را وارد کند و پس از تأیید، برنامه برای اجرا به گره اصلی دربرگیرنده‌ی موتور اسپارک فرستاده خواهد شد. همان‌طور که در بخش‌های قبلی اشاره شد، برای پیاده‌سازی الگوریتم‌ها از زبان اسکالا استفاده شده است. برای ارتباط بین رابط کاربری و موتور اسپارک، از یک REST Server آزاد و متن‌باز به نام لیوی[۱۱۰] بهره گرفته شده است [۱۹]. لیوی در گره اصلی اسپارک اجرا می‌شود و سپس با استفاده از متدهای اچ‌تی‌تی‌پی[۱۱۱] می‌توان ارتباط بین رابط کاربری و لیوی را برقرار کرد. لیوی خروجی‌های موردنظر از اجرای برنامه‌های کاربردی را در قالب اشیاء جِیسان[۱۱۲] به رابط کاربری باز می‌گرداند.

همچنین، یک تولید کننده‌ی جریان‌داده‌ی عددی برای استفاده در آزمون برنامه‌های کاربردی در این ابزار تعبیه شده است.

در شکل ۱۲، نمایی از رابط کاربری SDMiner نشان داده شده است.

---

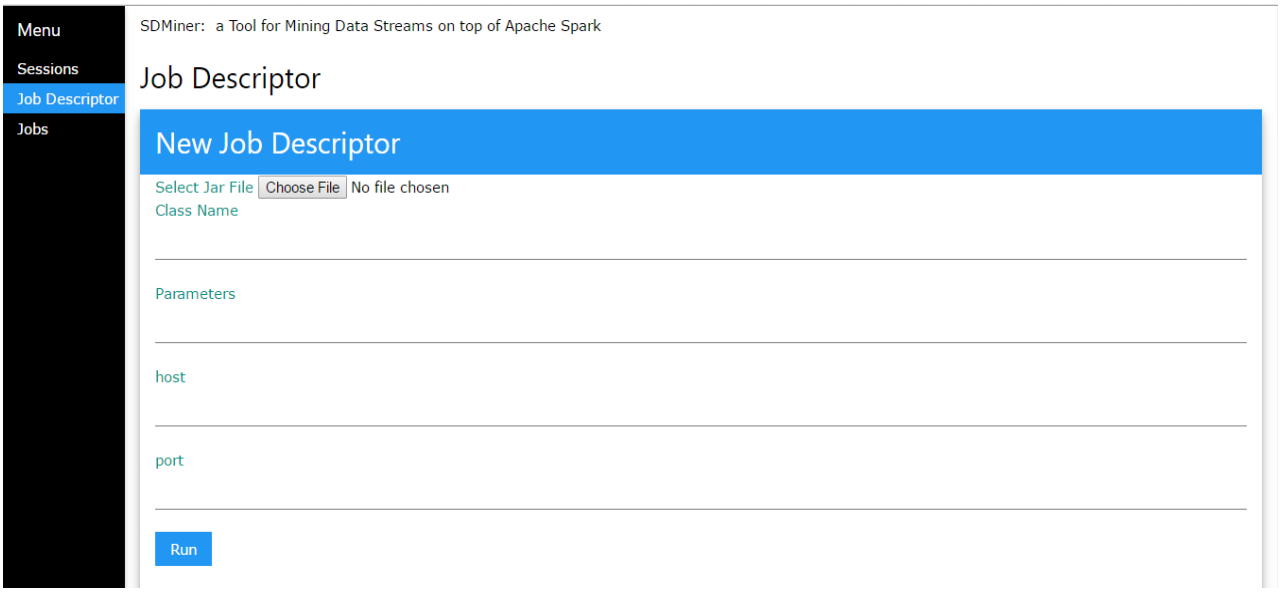
[۱۰۸] HTML

[۱۰۹] CSS

[۱۱۰] Livy

[۱۱۱] HTTP Methods

[۱۱۲] JSON Objects



شکل ۱۲- نمای تعریف عملیات داده‌کاوی در SDM ner

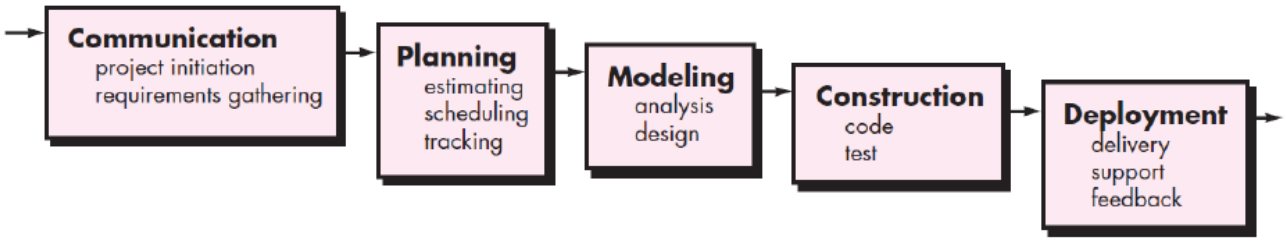
در بخش بعدی به فعالیت‌های تحلیل و طراحی صورت گرفته در این پروژه پرداخته خواهد شد.

۴.۲ تحلیل و طراحی نرم‌افزار

در این بخش به مدل فرآیندی مورد استفاده در این پروژه، و برخی از مستندات طراحی نرم‌افزار پرداخته خواهد شد.

۴.۲.۱ مدل فرآیندی آبشاری

از آنجا که در زمان تعریف پروژه، نیازمندی‌های نرم‌افزار ثابت و مشخص بوده‌اند، از مدل فرآیندی آبشاری[۱۱۳] برای پیاده‌سازی این پروژه استفاده شد. این مدل فرآیندی شامل پنج مرحله‌ی ارتباط[۱۱۴]، برنامه‌ریزی[۱۱۵]، مدل‌سازی[۱۱۶]، ساخت[۱۱۷]، و استقرار می‌باشد. این پنج مرحله در شکل ۱۱ نشان داده شده



شکل ۱۳- مدل فرآیندی آبشاری [۱۳]

است.

از جمله دلایل دیگر برای انتخاب این مدل فرآیندی، می‌توان به سادگی و قابل فهم بودن کل مدل و مراحل مختلف آن، و آسانی بررسی و کنترل مراحل مختلف اشاره کرد. این مدل یک روش خطیِ ترتیبی برای توسعه‌ی نرم‌افزار محسوب می‌شود. چون در این مدل، نرم‌افزار پس از یک‌بار پیمایش مراحل

---

[۱۱۳] Waterfall Process Model

[۱۱۴] Communication

[۱۱۵] Planning

[۱۱۶] Modeling

[۱۱۷] Construction

مختلف تولید می‌شود، باید نیازمندی‌های پروژه در ابتدا ثابت، مشخص و بدون ابهام باشند. در ادامه به فعالیت‌های صورت گرفته در راستای هر مرحله از این مدل اشاره می‌شود.

۴.۲.۱.۱ ارتباط

در این مرحله مطالعات و فعالیت‌هایی برای آشنایی با فضای مسأله، جمع‌آوری نیازمندی‌ها، آشنایی با کارهای مشابه، بررسی منابع مطالعاتی، و استفاده از نظرات استادان راهنما صورت گرفت.

۴.۲.۱.۲ برنامه‌ریزی

در این مرحله، برنامه‌ی زمان‌بندی انجام پروژه با توجه به نیازمندی‌ها و ضوابط گروه نرم‌افزار دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر صورت گرفت. برای راحت‌تر شدن زمان‌بندی، برای طراحی و پیاده‌سازی پیکرپاره‌های مختلف پروژه زمان‌های مشخصی تعیین شد. با توجه به سابقه‌ی انجام پروژه‌های دیگر، سعی شد تا از هر دو نوع برآوردهای خوش‌بینانه و بدبینانه اجتناب شده و زمان‌بندی مناسب بین این دو حالت مدنظر قرار گیرد.

۴.۲.۱.۳ مدل‌سازی

در این قسمت که مصادف با ارائه‌ی پیشنهاد پایان‌نامه[۱۱۸] به گروه نرم‌افزار دانشکده بود، با توجه به نیازمندی‌های پروژه، مستندات طراحی آماده شده و همچنین در مورد برخی فناوری‌های مورد استفاده در پروژه تصمیم‌گیری شد. این مستندات در بخش ۴,۳,۲ مرور شده‌اند.

۴.۲.۱.۴ ساخت

در این مرحله، با توجه به تحلیل‌های صورت گرفته در مرحله‌ی قبلی، مدل ارائه شده و همچنین فناوری‌های مورد استفاده، پیکرپاره‌های مختلف ابزار SDMiner پیاده‌سازی شدند. همچنین صحت عملکرد اجزای مختلف مورد آزمون قرار گرفت. برای آزمون صحت الگوریتم DRSFR، هم از استدلال‌های ریاضی و هم آزمون اجرای الگوریتم بر بستر اسپارک استفاده شد.

---

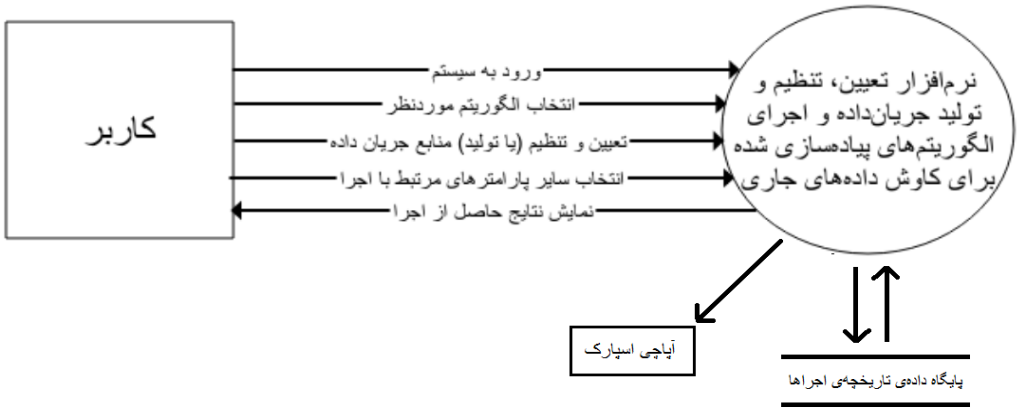
[۱۱۸] Thesis Proposal

۴.۲.۱.۵ استقرار

با نزدیک شدن به زمان ارائه‌ی نهایی پروژه، ابزار SDMiner به عنوان یک سامانه شامل پیکرپاره‌های مختلف پیاده‌سازی شده و هماهنگی و صحت عملکرد و ارتباط اجزای آن با یکدیگر بررسی شد. گزارش حاضر نیز به عنوان بخشی از مستندات راهنمای مربوط به این ابزار درنظر گرفته می‌شود.

۴.۲.۲ مستندات تحلیل و طراحی

در شکل ۱۴ نمودار مفهومی سطح صفر[۱۱۹] مربوط به این نرم‌افزار ارائه شده است.

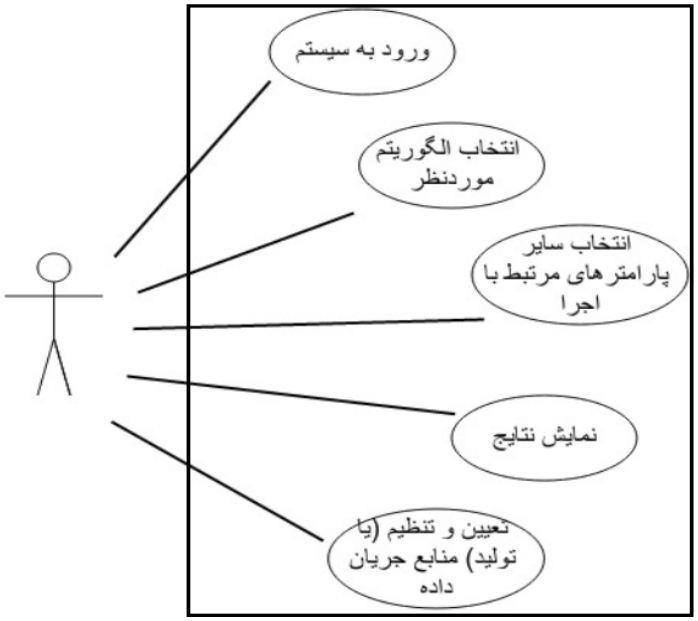


شکل ۱۴ - نمودار مفهومی سطح صفر

همچنین، در شکل ۱۵ نمودار مورد کاربرد[۱۲۰]، که مهم‌ترین نمودار در مرحله‌ی تحلیل سیستم است و مبنای طراحی قسمت‌های بعدی قرار می‌گیرد نشان داده شده است.

[۱۱۹] Context Diagram

[۱۲۰] Use Case Diagram



شکل ۱۵ - نمودار مورد کاربرد

۴.۳ خلاصه‌ی فصل

در این فصل به تحلیل، طراحی و پیاده‌سازی پروژه پرداخته شد. در ابتدا معماری لایه‌ای ابزار SDMiner معرفی شده و سپس پیکرپاره‌های مختلف آن مورد بررسی قرار گرفت. در ادامه، به مدل فرآیندی به کار رفته در این پروژه و برخی مستندات تحلیل و طراحی نرم‌افزار اشاره شد.

فصل بعدی به جمع‌بندی و کارهای آینده قابل انجام برای این پروژه خواهد پرداخت.

فصل پنجم -
جمع‌بندی و کارهای آینده

این گزارش به فرآیند منتهی به طراحی و پیاده‌سازی SDMiner، یک ابزار داده‌کاوی مبتنی بر آپاچی اسپارک برای داده‌های جاری اختصاص داشت. در ابتدا به داده‌های جاری، کاربردهای پردازش و کاوش آن‌ها در دنیای امروز، و چالش‌های این امر - که مسأله‌ی مورد بحث در این پروژه بود - پرداخته شد. پس از آن، مدلی کلاسیک برای پردازش داده‌های جاری معرفی شد. با توجه به خصوصیات منحصر به فرد جریان‌داده‌ها، پردازش داده‌های جاری در عمل نیاز به بسترهای توزیع‌یافته دارد، به همین در ادامه برخی از مطرح‌ترین بسترهای توزیع‌یافته‌ی پردازش داده‌های جاری مورد بررسی و مقایسه قرار گرفته و بستر آپاچی اسپارک برای استفاده در پیاده‌سازی این پروژه انتخاب شد. سپس به رابط برنامه‌نویسی کاربردی اسپارک‌استریمینگ و مدل برنامه‌نویسی آن پرداخته شد. پس از آن و در فصل سوم، مفهوم خلاصه‌سازی و نمونه‌برداری معرفی شده و الگوریتم نمونه‌برداری تصادفی با مخزن ثابت (RSFR) به عنوان یکی از الگوریتم‌های معمول نمونه‌برداری مورد بررسی قرار گرفت. با توجه به ماهیت غیرتوزیع‌یافته‌ی این الگوریتم، به طراحی و پیاده‌سازی نسخه‌ی توزیع‌یافته و مبتنی بر مدل برنامه‌نویسی اسپارک‌استریمینگ این الگوریتم (DRSFR) پرداخته شد. در ادامه و در فصل چهارم، در مورد تحلیل، طراحی، پیاده‌سازی و ارزیابی قسمت‌های مختلف SDMiner و مدل فرآیندی به کار رفته در طول انجام پروژه بحث شد.

ابزار تولید شده تا بدینجای کار، نیازمندی‌های مطرح شده برای پروژه را برآورده می‌کند. با وجود این موضوع، همچنان می‌توان آن را از جنبه‌های مختلف گسترش داد. در ادامه، به موارد و جهت‌گیری‌هایی برای کارهای بیشتر بر روی این پروژه اشاره می‌گردد.

- مهم‌ترین موردی که می‌توان با تمرکز بر آن، این ابزار را بهبود بخشید، پیاده‌سازی الگوریتم‌های بیشتر و در واقع اغنای کتابخانه‌ی الگوریتم‌های کاوش داده‌های جاری است. در انتخاب الگوریتم‌ها باید به این نکته توجه داشت که موازی‌سازی و تبدیل الگوریتم در قالب مدل برنامه‌نویسی اسپارک‌استریمینگ، اساسی‌ترین بخش کار است.
- رابط گرافیکی را می‌توان با توجه به بازخورد دریافتی از کاربران، بهبود داد. همچنین، در حال حاضر، نمایش خروجی پس از درخواست کاربر صورت می‌گیرد و می‌توان فرآیندی برای نمایش اتوماتیک خروجی در بازه‌های مشخص تعبیه کرد. از طرف دیگر، با ازدیاد حجم داده‌ها و استفاده

از الگوریتم‌های پیچیده‌تر نیاز به استفاده از روش‌های مصورسازی[۱۲۱] برای نمایش نتایج، بیشتر احساس می‌شود.

- در کاربردهای واقعی، لازم است موتور آپاچی اسپارک بسته به برنامه‌ی کاربردی مورد استفاده و محیط اجرا، میزان‌سازی[۱۲۲] شود. می‌توان امکانی به این ابزار اضافه کرد تا کاربر بدون نیاز به مراجعه به گره اصلی، از طریق همین ابزار و در قالب گرافیکی بتواند به میزان‌سازی موتور اسپارک بپردازد.
- می‌توان پشتیبانی از دریافت و پردازش هم‌زمان جریان‌داده‌های مختلف را به این ابزار اضافه کرد.
- برای افزایش جامعیت ابزار، می‌توان پشتیبانی از بسترهای توزیع‌یافته‌ی دیگر را هم به آن افزود.

لازم به ذکر است که تمامی مستندات و کدهای پروژه برای دسترسی عمومی و آزاد در پوشه‌ی مخصوص پروژه در گیت‌هاب[۱۲۳] قرار داده خواهد شد.

[۱۲۱] Visualization

[۱۲۲] Tuning

[۱۲۳] http://github.com/ssheikholeslami

منابع و مراجع

[۱] Aggarwal, Charu C. Data streams: models and algorithms. Vol. 31. Springer Science & Business Media, 2007.

[۲] Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive datasets. Cambridge University Press, 2014.

[۳] Andrade, Henrique CM, Buğra Gedik, and Deepak S. Turaga. Fundamentals of Stream Processing: Application Design, Systems, and Analytics. Cambridge University Press, 2014.

[۴] Han, Jiawei, Micheline Kamber, and Jian Pei. Data mining: concepts and techniques. Elsevier, 2011.

[۵] "Apache Flink: Scalable Batch and Stream Data Processing." Web. 31 Jan. 2016. <http://flink.apache.org/>.

[۶] "Apache Storm." Web. 31 Jan. 2016. <http://storm.apache.org/>.

[۷] Zaharia, Matei, et al. "Discretized streams: Fault-tolerant streaming computation at scale." Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013.

[۸] Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[۹] “Apache Spark™ - Lightning-Fast Cluster Computing." Web. 31 Jan. 2016. <http://spark.apache.org/>.

[۱۰] "BDAS, the Berkeley Data Analytics Stack." AMPLab UC Berkeley. Web. 31 Jan. 2016. <http://amplab.cs.berkeley.edu/software/>.

[۱۱] "Spark Streaming | Apache Spark." Web. 31 Jan. 2016. <http://spark.apache.org/streaming/>.

[۱۲] Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." NetDB, 2011.

[۱۳] Pressman, Roger S. Software engineering: a practitioner's approach., 7th Edition, McGraw-Hill, 2009.

[۱۴] Kamburugamuve, Supun, and Geoffrey Fox. "Survey of Distributed Stream Processing.", 2015.

[۱۵] Bifet, Albert, et al. "StreamDM: Advanced Data Mining in Spark Streaming." 2015 IEEE International Conference on Data Mining Workshop (ICDMW). IEEE, 2015.

[۱۶] "Spark Streaming Programming Guide." Spark Streaming. Web. 06 Apr. 2016. <http://spark.apache.org/docs/latest/streaming-programming-guide.html>.

[۱۷] Das, Tathagata, Matei Zaharia, and Patrick Wendell. "Diving into Spark Streaming's Execution Model." Databricks. 2015. Web. 06 Apr. 2016. <https://databricks.com/blog/2015/07/30/diving-into-spark-streamings-execution-model.html>.

[۱۸] "MLlib | Apache Spark." Web. 06 Apr. 2016. <http://spark.apache.org/mllib/>.

[۱۹] "Livy, an Open Source REST Service for Apache Spark" Web. 21 June. 2016. <http://livy.io/>.

پیوست

بخش‌هایی از پیاده‌سازی

DRSFRWithState.scala

```
import java.util.Random

import org.apache.log4j.{Level, Logger}
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StateSpec, StreamingContext, State}

/**
  * Created by sinash on 5/27/16.
  */
object DRSFRWithState {
  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: <source hostname> <source port> <interval
duration> <reservoir size>")
      System.exit(1)
    }
    val reservoirSize = args(3).toInt
    var sampleSet = new Array[Long](reservoirSize)
    Logger.getLogger("org").setLevel(Level.OFF) //disable logging
    val conf = new SparkConf().setAppName("DRSFR-WithState")
    val ssc = new StreamingContext(conf, Seconds(args(2).toLong))
    ssc.checkpoint("_checkstatepoint")
    val stream = ssc.socketTextStream(args(0), args(1).toInt)
    val parsedStream = stream.map( elem => ("sharedKey", elem.toInt))
    val initialRDD = ssc.sparkContext.parallelize(List(("sharedKey", 0)))
    val mappingFunc = (sharedKey: String, element: Option[Int], globalIndex:
State[Int]) => {
      val newGlobalIndex = globalIndex.getOption.getOrElse(0) + 1
      val output = (element.getOrElse(0), newGlobalIndex)
      globalIndex.update(newGlobalIndex)
      output
    }
    val indexedStream =
parsedStream.mapWithState(StateSpec.function(mappingFunc).initialState(initialR
DD))
```

```
    indexedStream.print()
    // to start the processing
    ssc.start()
    ssc.awaitTermination() // wait for the computation to terminate
  }
}
```

RandomNumberGenerator.scala

```
import java.io.PrintWriter
import java.net.ServerSocket
import scala.util.Random

/**
  * Created by sinash on 5/12/16.
  * Based on example code from "Machine Learning with Spark" book by Nick
Pentreath, Packt Publishing
  */
object RandomNumberGenerator {
  var counter = 1

  def main(args: Array[String]) {
    val random = new Random()
    if (args.length < 4) {
      System.err.println("Usage: <port> <max value> <events per second>
<indexed?: y/n>")
      System.exit(1)
    }
    //create a network producer
    val listener = new ServerSocket(args(0).toInt)
    println("listening on 9999")
    var sleepTime = 100 //default, 10 events per second
    if (args(2).toInt != 0) {
      sleepTime = 1000 / args(2).toInt
    }
    if(args(3).toString.equalsIgnoreCase("y")) {
      while (true) {
```

```
    val socket = listener.accept()
    counter = 1
    new Thread() {
      override def run = {
        println("client connected from: " + socket.getInetAddress)
        val out = new PrintWriter(socket.getOutputStream(), true)

              while (true) {

          Thread.sleep(sleepTime)
          val num = random.nextInt(args(1).toInt)
          out.print(num + " " + counter)
          counter = counter + 1
          out.write("\n")
          out.flush()
        }
        socket.close()
      }
    }.start()
  }
}
else{
  var num = 1
  while (true) {
    val socket = listener.accept()
    new Thread() {
      override def run = {
        println("client connected from: " + socket.getInetAddress)
        val out = new PrintWriter(socket.getOutputStream(), true)
        while (true) {
          Thread.sleep(sleepTime)
          num += 1
          out.print(num)
          out.write("\n")
          out.flush()
        }
        socket.close()
```

```
          }
        }.start()
      }
    }
  }
}
```

Amirkabir University of Technology
(Tehran Polytechnic)

Department of Computer Engineering and Information Technology

B.Sc. Thesis in Software Engineering

Title

Implementing a Tool for Mining Data Streams on Top of Apache Spark

By
Sina Sheikholeslami

Advisors
Dr. Amir H. Payberah
Dr. Seyyed Rasool Moosavi

June 2016