پایان‌نامه کارشناسی ارشد

گروه مهندسی کامپیوتر

ارائه یک روش شاخص‌گذاری مقیاس‌پذیر و مبتنی بر موجودیت روی داده‌های RDF

نگارنده:

فاطمه عبیری

استاد راهنما:

دکتر محسن کاهانی

بهمن ماه ۱۳۹۲

چکیده

حضور وب معنایی و استقبال گسترده از آن در سال‌های اخیر، باعث شده‌است که توسعه‌دهندگان وب، تلاش‌های بسیاری را تا به امروز در جهت سازماندهی این داده‌ها انجام دهند. اما سیستم‌های شاخصگذاری ارائه شده هنوز در سازماندهی این داده‌ها با کاستی‌هایی روبه‌رو هستند. این امر به این دلیل است که روش‌هایی که برای سازماندهی داده‌های وب معنایی ارائه می‌شوند، باید از مقیاس‌پذیری لازم، هم در ذخیره و هم در بازیابی داده‌ها برخوردار باشند.

برای این منظور روشی برای سازماندهی این داده‌ها در یک سیستم شاخص‌گذاریRDF پیشنهاد شده‌است. در یک مجموعه داده‌ی RDF، اسناد وب معنایی در عین نیمه‌ساخت‌یافته بودن، دارای نظم خاصی در تعریف موجودیت‌ها هستند. همچنین طبق آمارهای اخیر، اکثر کاربران وب نیز در پرس‌وجوها به دنبال یک موجودیت با مجموعه صفاتی مشخص هستند. بنابراین در روش پیشنهادی، یک الگوریتم خوشه‌بندی، جهت خوشه‌بندی موجودیت‌های مشابه، ارائه شده‌است. با توجه به اینکه جنبه‌ی عملیاتی یک سیستم شاخص‌گذاریRDF، مقیاس‌پذیری آن می‌باشد، از پایگاه‌داده‌های NoSQL جهت شاخص‌گذاری هر دسته از موجودیت‌های مشابه، استفاده شده‌است. نتایج نشان می‌دهد که این سیستم توانسته است در مقابل حجم یک مجموعه داده‌ی کامل RDF موفق عمل کند.

فهرست مطالب

فصل ۱- مقدمه

۱-۱- تعریف مساله

با توجه به گسترش روزافزون منابع اطلاعاتی مختلف در دنیای وب، نگهداری و بازیابی این اطلاعات گسترده برای استفاده کاربران، مسئله‌ی مهم و قابل توجهی به شمار می‌آید. به همین منظور توسعه-دهندگان وب چهارچوبی را جهت بازیابی اطلاعات[۱]تعریف نمودند تا گام‌های اجرای عملیات روی داده‌ها بطور کامل روشن شود. بازیابی اطلاعات درواقع به مجموعه‌ای از عملیات ارائه(انتشاراطلاعات)، ذخیره‌سازی، سازماندهی(شاخص‌گذاری) و دسترسی به اقلام اطلاعاتی گفته می‌شود[Del10]. هدف از بازیابی اطلاعات کمک به کاربر جهت دسترسی به اطلاعات مورد نیازش است. این اطلاعات توسط کاربر مشخص و به سیستم بازیابی اطلاعات جهت پردازش ارسال می‌شود. با ارسال یک پرس‌وجو، سیستم بازیابی اطلاعات، اقلام اطلاعاتی را که با پرس‌وجوی کاربر مطابقت دارد، بازیابی می‌کند.

موتورهای جستجوی سنتی به عنوان یک سیستم بازیابی اطلاعات، در مقابل پرس‌وجوی کاربر، اسناد مرتبط را بازیابی می‌کنند. اقلام اطلاعاتی که این سیستم‌ها باید در جواب درخواست کاربر به آن بپردازند، اسناد است. این اسناد معمولا، متنی ارائه می‌شوند و درواقع سند از دید سیستم، مجموعه‌ای از کلمات متوالی بدون ساختار است[Mel01].

فرایند بازیابی در این سیستم‌ها به این شکل است که کاربر یک درخواست را در قالب کلمات کلیدی در سیستم وارد می‌کند. سپس پرس‌وجو تجزیه می‌شود و جهت بازیابی اسناد مرتبط،

---

[۱]-Information retrieval

پردازش‌هایی صورت می‌گیرد. پردازش پرس‌وجو توسط یک ساختار شاخص‌گذاری[۱] مانند شاخص‌گذاری معکوس[۲] جهت محاسبه‌ی سریع اطلاعات حمایت می‌شود. در این نحوه شاخص‌گذاری، به ازای هر کلمه‌ی درون اسناد که قابل جستجو باشد، لیستی از اسنادی که آن کلمه در آن اسناد آمده‌است نگهداری می‌شود که به این ساختار، شاخص‌گذاری معکوس می‌گویند[Nar09]. مشکلی که در این نوع سیستم‌های بازیابی اطلاعات وجود دارد عدم در نظر گرفتن ساختار داده‌ها است. بنابراین ظهور وب معنایی ودر نتیجه‌ی آن حضور داده‌های نیمه‌ساخت یافته[۳] باعث شد که ساختار داده‌ها نیز مورد اهمیت واقع شود[Del10].

وب معنایی چهارچوبی است که اجازه‌ی اشتراک داده‌ها در برنامه‌های کاربردی، موسسات و سازمانها را می‌دهد و بر پایه‌ی چهارچوب توصیف منابع[۴] و به اختصار RDF پایه‌ریزی شده‌است[Sem].

RDF چهارچوبی است که امکان قابل استفاده بودن[۵] قابل کشف بودن و دسترسی آسان را به داده‌ها می‌دهد و از همه مهم‌تر قابلیت تجمیع[۶] از چندین منابع ناهمگون را داراست. برخلاف وب سنتی که بین اسناد روابط معنایی وجود ندارد، RDF امکان ایجاد پیوندها و روابط معنادار را بین مفاهیم، ایجاد می‌کند. مهمترین مزیت RDF در وب معنایی سادگیِ مدل داده آن است که امکان ارائه‌ی یک تعریف

۱-Indexing

۲-Inverted Index

۳- Semi-Structured data

۴- Resource Description Framework

۵- Re-usable

۶- Integration

رسمی‌از معنا را به کابران وب می‌دهد. بنابراین با استفاده از این تعریف رسمی، امکان استنتاج[۱]هم مهیا می‌شود. مدل داده RDF انعطاف‌پذیری بالایی برای نمایش داده‌ها با شِما یا ساختارهای مختلف دارد. مدل داده RDF یک مجموعه از سه‌گانه‌ها است که هر یک دارای نهاد، مسند و گزاره‌است[Fun12].

وب امروزه در حال تجربه حجم زیادی از داده‌های RDF است. آخرین آمارها نشان می‌دهد که کش این داده‌ها به حدود ۵۲۰۰۰ ملیون سه‌گانه رسیده‌است. بنابراین این حجم بسیار زیاد از داده‌ها نیاز به یک روش مقیاس‌پذیر، جهت شاخص‌گذاری RDF دارد[LOD].

یک سیستم شاخص‌گذاری RDF از دو بخش ذخیره و بازیابی داده تشکیل می‌شود. با توجه به حجم روزافزون داده‌های RDF، مهمترین انتظاری که از یک سیستم شاخص‌گذاری می‌رود مقیاس‌پذیری آن، هم در ذخیره و هم در بازیابی داده‌ها است. این سیستم‌ها باید در عین ذخیره‌ی داده‌های حجیم RDF، با ارائه یک واسط کاربری مناسب برای پرس‌و جوهای پیشرفته، امکان دسترسی به چندین منبع داده را به کاربران در کمترین زمان ممکن بدهند.

بنابراین دو چالش اساسی در پیاده سازی یک سیستم مقیاس‌پذیرِ شاخص‌گذاری RDF مطرح می‌شود. چالش اول، انتخاب نوع سیستمی‌است که برای ذخیره و مدیریت شاخص استفاده می‌شود و چالش دوم، انتخاب نوع شِما (مدل داده) که برای ذخیره و بازیابی داده‌های RDF به کار گرفته می‌شود. در ادامه شرح مختصری از هریک از این چالش‌ها بیان می‌شود.

برای رفع چالش سیستم ذخیره و مدیریت شاخص، گزینه‌های مختلفی وجود دارد. گزینه‌ی اول طراحی یک سیستم شاخص‌گذاری محلی[۲]متناسب با ساختار داده‌ی RDF است[Del10]. اما ارائه‌ی

۱- Resoning

۲-Native

چنین سیسمتی نیاز به مهارت بالا در مسائلی همچون اصول طراحی پایگاه‌داده‌ها و مباحث توزیع شده دارد. بنابراین طراح درگیر مسائلی خارج از مسئله اصلی یعنی وب‌معنایی شده و باید تمام مراحل پیاده سازی یک پایگاه‌داده را مجدد طی کند[Owe08] [Neu10][Del10]. دومین گزینه برای ذخیره‌سازی داده‌ها، استفاده از پایگاه‌داده‌های رابطه‌ای است که یکی از معمول‌ترین روش‌های ذخیره‌ی این نوع داده‌ها بشمار می‌آید. اگر چه این سیستم‌ها عملکرد خوب و کارایی برای حجم داده‌های زیاد دارند اما مقیاس-پذیری آنها محدود به اندازه‌ای معین با هزینه سخت‌افزاری بسیار بالا است[RDB]. در واقع این سیستم‌ها امکان مقیاس‌پذیری افقی را با پیچیدگی بسیار بالایی ارائه می‌دهند[Med11]. از این رو سیستم‌های NOSQL [NOSQL] به منظور حل محدودیت مقیاس‌پذیریِ پایگاه‌داده‌ای رابطه‌ای ظهور کردند. این سیستم‌ها با موفقیت در مقیاس بزرگی از اینترنت در شرکت‌هایی همچون Google[cha06]، Facebook[Fun12] و Amazoon[Bug12] با موفقیت مورد استفاده قرار گرفتند. مهمترین ویژگی پایگاه داده‌های NOSQL کد باز بودن، مقیاس‌پذیری افقی و امکان پردازش داده‌ها در فضای توزیعی است. این سیستم‌ها معمولا پشتیبانی کامل از ACID را فدای عملکرد بالا نموده اند.

چالش دوم بعد از انتخاب سیستم ذخیره و مدیریت شاخص‌ها، طراحی نوع شِما و ساختاری است که داده‌های RDF طبق آن مدل داده، ذخیره می‌شوند. روش‌های مختلفی برای تعریف نوع شِمای ذخیره داده‌های RDF ارائه شده‌اند که هریک نقطه‌ی ضعف و قوت مربوط به خود را دارند. شِماها از جنبه‌های مختلفی مورد ارزیابی قرار می‌گیرند. مهمترین جنبه‌ای که برای ارزیابی یک شِما باید به آن دقت شود، عملکرد شِمای مورد نظر در پوشش الگوهای دسترسی به داده‌ها است. طبق آمارهایی که در[Gall11]منتشر شده‌است حدود ۹۹,۹۷٪ از پرس‌وجوهایی که از طریق کاربران وب به سیستم اعمال شده‌است، پرس‌وجوهایی هستند که در آن کاربران وب به دنبال یک موجودیت با مجموعه‌ای از صفات هستند. به این نوع پرس‌وجوها، پرس‌وجوهای مبتنی بر موجودیت گویند. بنابراین پوشش این نوع

دسترسی، آن هم با سرعت بالا می‌تواند به بازدهی یک سیستم شاخص‌گذاری RDFکمک موثری کند. سیستم‌هایی که قصد حمایت از همه نوع الگوی دسترسی را دارند، طبیعتا از سیستم‌هایی که تنها تمرکزشان روی یک نوع الگوی دسترسی، همچون دسترسی مبتنی بر موجودیت است، بهتر عمل نخواهند کرد. چرا که که مدل داده‌ی سیستم‌های متمرکز روی یک پرس‌وجو، مدل‌داده‌ی مخصوص به خود را دارد و تمام بهینه سازی‌های مربوط به ذخیره و بازیابی داده‌ها نیز بر اساس همین مدل داده انجام می‌شود.

روش‌هایی که تمرکز اصلی آنها، پوشش الگوی دسترسی مبتنی بر موجودیت است، دارای چالش-هایی از جمله مقیاس‌پذیری هستند. روش [Wil06] شِمایی همچون شکل(۱-۱) را ارائه داده است.

Property Table

Subj.	Type	Title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL
ID5	CDType	"GHI"	"1995"
ID6	BookType	NULL	"2004"

Left-Over Triples

Subj.	Prop.	Obj.
ID1	author	"Fox, Joe"
ID2	artist	"Orr, Tim"
ID2	language	"French"
ID3	language	"English"

شکل ۱-۱: ارائه‌ی یک شِما جهت حمایت از پرس‌وجوهای مبتنی بر موجودیت[Lev09]

در این نوع ذخیره‌سازی، چندین صفت به ستون‌های یک جدول نگاشت داده می‌شوند. بنابراین نیاز به الحاق جداول برای بازیابی یک موجودیت، بسیار کاهش می‌یابد. اما همانطور که در این شِما مشهود است، وجود مقادیر *NULL* برای جداول رابطه‌ای یک سربار محسوب می‌شود. همچنین صفاتی که کمتر با بقیه رخ داده‌اند و باعث مقادیر *NULL* زیاد در جدول می‌شوند، در جدول دیگر همچون جدول سه‌گانه، ذخیره می‌شوند. البته این جدا شدن صفات باعث شده‌است که الحاق به طور کامل در بازیابی یک موجودیت حذف نشود. این شِما دارای مزایایی از جمله موارد زیر است.

پشتیبانی موثر از پرس جوی مبتنی برموجودیت- تمرکز این نوع شِما حمایت از پرس‌وجوی مبتنی بر موجودیت است که هرچند در پشتیبانی از همه نوع پرس‌وجو، عملکرد ضعیف‌تری دارد ولی به پرس‌وجوی رایج مبتنی بر موجودیت سریعتر از باقی جواب می‌دهد[Del10][Lev09].

سرعت در الحاق داده‌ها- **به** دلیل ذخیره‌ی صفات هر موجودیت در مجاورت هم، الحاق‌های subject-subject در این نوع ذخیره‌سازی به نحو موثری کاهش می‌یابد.

اندازه شاخص- اندازه شاخص در این شِما به نحو موثری کاهش یافته‌است. این کاهش به دلیل عدم تکرار موجودیت‌ها به ازای یک صفت خاص است.

هزینه‌ی بروزرسانی شاخص- در بروزرسانی اطلاعات یک موجودیت، سرعت به نسبت بالاست. به این دلیل که صفات یک موجودیت به همراه آن ذخیره شده‌است.

اما این روش شاخص‌گذاری به دلیل پیاده‌سازی در جداول ربطه‌ای [Levo9] با چالش‌هایی روبه رو است[Aba07] :

حضور مقادیر NULL و نیاز به الگوریتم موثر برای دسته بندی داده‌ها – در طراحی این نوع شِما، غیرساخت‌یافته بودن داده‌ها باعث شده تا تقابلی بین تعداد جداول ایجاد شده و مقادیر *NULL* صورت گیرد که باز هم حضور مقادیر *NULL* در جداول رابطه‌ای غیرقابل انکار است و مقیاس‌پذیری آن را در حجم داده‌های بالا تحت تاثیر قرار می‌دهد[Lev09].

عدم پشتیبانی از صفت‌های چند مقداری- از صفت‌های چند مقداری به شکل بسیار ضعیفی پشتیبانی می‌کند. منظور از صفت‌های چند مقداری، صفاتی است همچون ایمیل که ممکن است یک فرد دارای چند ایمیل باشد.

حضور پرس‌وجوهایی که predicate آنها متغیر باشد و ذکر نشده باشد- بازیابی اطلاعات در این نوع شِما دچار پیچیدگی می‌شود چرا که اطلاعاتی کامل، از موجودیت‌های درون پایگاه‌داده نیست تا بتوان جدول مربوطه را حدس زد.

عدم حذف کامل الحاق در بازیابی یک موجودیت- با توجه به اینکه یک سری از Subjectها به دلیل حذف بخشی از سربار *NULL* در دو جدول قرار می‌گیرند، بنابراین در بازیابی یک موجودیت عمل الحاق وجود خواهد داشت.

مقیاس‌پذیری- به دلیل نوع الگوریتم انتخابی در طراحی این نوع شِما و نیز پایگاه‌داده‌ی رابطه‌ای مورد استفاده، این شِما مقیاس‌پذیر نخواهد بود.

۲-۱- راه حل پیشنهادی

با توجه به اهمیت و محبوبیت پرس‌وجوهای مبتنی بر موجودیت، تمرکز اصلی روش پیشنهادی نیز پشتیبانی ویژه از این نوع پرس‌وجوها است. در روش‌های پیشین، چالش اصلی مطرح شده در شِماهای مبتنی بر موجودیت عدم مقیاس‌پذیری سیستم‌ها به دلیل استفاده از پایگاه داده‌های رابطه‌ای است. بنابرین هدف از راه حل پیشنهادی ارائه‌ی یک روش شاخص‌گذاری مقیاس‌پذیر و مبتنی بر موجودیت روی داده‌هایRDF است. جهت برآورده کردن امکان مقیاس‌پذیری، با توجه به مطالب گفته شده در بخش(۱-۱)، این روش‌شاخص‌گذاری روی یک پایگاه‌داده‌ی NOSQL ارائه می‌شود.

هدف اصلی این تحقیق پیداکردن روشی موثر برای ذخیره‌سازی داده‌های RDF است به نحوی که بتوان عملکرد مطلوبی در ذخیره و بازیابی این نوع داده‌ها داشت. حجم گسترده‌ای از انواع سیستم‌های

NOSQL از جمله پایگاه‌داده‌ی مبتنی بر گراف، مخزن ذخیره‌سازی مبتنی بر سند، مخزن کلید-مقدار[۱] و مخزن ستون‌های زیاد(ستون‌های خانواده[۲]) متولد شده‌اند[NOSQL]. از بین این سیستم‌ها باید بهترین سیستم را برای ذخیره‌ی RDFها شناسایی کرد که برای این منظور باید ببینیم که از یک سیستم ذخیره RDF وجود چه ویژگی‌هایی انتظار می‌رود. سیستم انتخابی اولا باید قابلیت انطعاف‌پذیری بالایی در ارائه‌ی شِمای مورد نظر داشته باشد که بتوان به راحتی ساختار مدل داده‌ی RDF و شِمای مبتنی بر موجودیت را به آن نگاشت داد. دوما با توجه به اینکه شاخص مربوطه برای انجام اعمال محاسباتی نیاز به تجمیع با یک چهارچوب محاسباتی توزیعی دارد، بنابراین سیستم مورد نظر باید این امکان را فراهم آورد. دربین گزینه‌های بالا HBase[۳] [Med11] با ذخیره‌ی داده‌ها براساس ستون‌های زیاد(ستون‌های خانواده) بهترین گزینه برای ذخیره داده‌ها است. این سیستم انعطاف‌پذیری بالایی در طراحی شِما دارد، کد باز است و از حمایت توسعه‌دهندگان برخوردار است[Hbase]. این سیستم در مقیاس گسترده مثل سیستم پیام‌دهی Facebook که خوشه‌ی توزیعی آن تا هزاران گره رسیده، استفاده شده‌است[Fun12]. HBase سازگاری[۴] را در حد بالا تامین می‌کند که در نتیجه‌ی آن، داده‌ها هنگام خواندن همواره بروز هستند. علاوه بر این دسترس پذیری داده‌ها[۵]با روش تکثیرِ[۶]اسناد، توسط Hadoop [Hadoop]که در زیرساخت HBase قرار

۱- KeyValue Store

۲-Column Family

۳-Hadoop DataBase

۴-Consistency

۵- Availability

۶- Replication

می‌گیرد، برقرار است. همچنین امکان انجام پردازش‌های توزیعی نیز با استفاده از Hadoop امکان‌پذیر شده‌است.

با استفاده از HBase چالش‌هایی از جمله: مقیاس‌پذیری، عدم پشتیبانی از صفت‌های چند مقداری و حضور مقادیر *NULL* در این نوع شِما حل می‌شود. شِمایی که جهت رفع این چالش‌ها روی HBase ارائه شد شِمای مبتنی بر جدول تکی است[empri]. اما در روش پیشنهادی، شِمایی متفاوت ارائه شده-است. در این شِمای جدید با استفاده از یک الگوریتم خوشه‌بندی روی صفات، چالش‌های شِمای ارائه شده‌ی در جداول رابطه‌ای [Levo9] از جمله حضور پرس‌وجوهایی که predicate آنها متغیر باشد و عدم حذف کامل الحاق در بازیابی یک موجودیت، حل شده‌است. و همچنین نرخ بازیابی داده‌ها نسبت به شِمای تک جدولی ارائه شده روی HBase [empri] بیشتر شده‌است.

در روش پیشنهادی، ابتدا الگوهایی از صفات که دائما به ازای هر موجودیت ظاهرشده‌اند، کشف می‌شوند. سپس موجودیت‌های با الگوی مشابه با استفاده از یک فرمول معیار شباهت، خوشه‌بندی و در پایگاه‌داده‌ی مقیاس‌پذیر و توزیعی HBase ذخیره می‌شوند. در مرحله‌ی بازیابی با استفاده از این خوشه‌ها که به نوعی ساختار یک مجموعه‌داده‌ی RDF[۱] را بطور خلاصه بیان می‌کنند، موجودیت‌هایی با الگویی از صفات که مطابق با پرس‌وجوی اعمالی به سیستم هستند، بازیابی می‌شوند. همچنین با استفاده از اطلاعات به دست‌آمده از ساختار موجودیت‌های درون مجموعه‌داده، می‌توان به کاربران هم در اجرای پرس‌وجوهای RDF و هم در تعریف یک موجودیت روی مجموعه‌داده‌ی مورد نظر کمک کرد.

۳-۱- نوآوری‌های راه‌حل پیشنهادی

نوآوری سیستم پیشنهادی در موارد زیر خلاصه می‌شود:

---

[۱]-DataSet

۱- ارائه‌ی الگوریتم خوشه‌بندی موجودیت‌ها جهت ارائه یک شِمای شاخص‌گذاری RDF و در نتیجه حذف کامل الحاق در بازیابی یک موجودیت

۲- ارائه‌ی یک معیار شباهت جهت اندازه‌گیری میزان شباهت بین موجودیت‌های یک مجموعه‌داده

۳- ارائه‌ی یک شِمای مبتنی بر موجودیت، متناسب با قابلیت‌های پیشرفته‌ی HBase

۴- پیشنهاد صفات به کاربر با توجه به داده‌های واقعی در جداول

۴-۱- ساختار پایان‌نامه

سازماندهی مطالب پایان‌نامه در این بخش شرح داده می‌شود. در فصل بعد ابتدا مفاهیم پایه مورد نیاز جهت پیاده‌سازی سیستم پیشنهادی شرح و سپس به معرفی سیستمهای مرتبط با سیستم پیشنهادی پرداخته می‌شود. در فصل سوم، راه حل پیشنهادی برای شاخص‌گذاری داده‌های RDF توضیح داده می‌شود. در فصل چهارم، جزئیات پیاده‌سازی سیستم پیشنهادی، نحوه‌ی ارزیابی و نتایج حاصل از ارزیابی آورده شده‌است و در نهایت، فصل پنجم به نتیجه‌گیری و کارهای آینده اختصاص دارد.

فصل ۲- مرور ادبیات

سیستم پیشنهادی این پایان‌نامه یک سیستم شاخص‌گذاری مبتنی بر موجودیت RDF است. هدف از ارائه این سیستم حل چالش‌هایی است که روش‌های پیشین با آن رو به رو هستند. بنابراین در این فصل ابتدا در بخش (۱-۲) به تشریح مفاهیم پایه‌ای پرداخته می‌شود که پیش‌نیاز اولیه‌ی طراحی سیستم پیشنهادی هستند. سپس در بخش (۲-۲) به مرور کارهای مرتبط با سیستم پیشنهادی پرداخته می‌شود.

۱-۲- مفاهیم پایه

در این بخش به شرح مفاهیم پیش‌نیاز جهت درک راهکارهای پیشین در شاخص‌گذاری داده های RDF می‌پردازیم.

RDF -۲-۱-۱

داده‌ها در وب معنایی به فرم سه‌گانه< نهاد,مسند,گزاره> هستند، در لاتین این داده‌ها به شکل <Subject, Predicate , Object> نمایش داده می‌شوند. یک مجموعه سه‌گانه می‌تواند به عنوان گراف‌هایی برچسب دار نمایش داده شود به طوری که گره‌ها، موجودیت (نهاد و یا گزاره‌های در قالب منبع) و لبه‌ها، صفت(مسند) هستند.[۱]

استانداردی که برای زبان پرس‌وجو روی داده‌های RDF مطرح می‌شود SPARQL[۲] است، که بر پایه‌ی تطابق الگوی گراف، جهت برقراری امکان پرس‌وجو روی ساختار داده‌ها عمل می‌کند.

---

[۱]-Http://www.w3.org/RDF/

[۲]-SPARQL query language: www.w3.org/TR/sparql11-query/

۲-۱-۲- مدل برنامه‌نویسی نگاشت-کاهش

مدل برنامه‌نویسی نگاشت-کاهش[۱] جهت پردازش و تولید مجموعه‌داده‌های بزرگ در فضای توزیعی بکار می‌رود. در این مدل برنامه‌نویسی، برنامه‌ی کاربر باید دارای دو تابع اساسی نگاشت و کاهش باشد. عملیات هر کدام ازاین دو تابع روی جفت کلید-مقدار است. در مرحله نگاشت، یک سری از کلید-مقدارها از فایلهای ورودی ساخته و عملیات مورد نظر کاربر روی این داده‌ها انجام می‌شود. سپس اگر محاسبات در مرحله‌ی نگاشت کافی بود داده‌ها در قالب کلید-مقدار در خروجی چاپ می‌شوند. اما اگر نیاز به پردازش-های بیشتری بود، این داده‌ها در قالب جفت کلید-مقدار به مرحله‌ی کاهش فرستاده می‌شود. در مرحله‌ی کاهش، کاربر جفت کلید-مقدار را به صورت مرتب شده (بر اساس کلیدها) در اختیار دارد که پردازش‌های مورد نظر را روی آن انجام می‌دهد[Had].

۲-۱-۳- Hadoop

Hadoop یک پیاده‌سازی کدباز از چهار چوب مدل برنامه‌نویسی توزیعی نگاشت-کاهش است. این سیستم در پی پروژه‌ی موفق فایل سیستم توزیع شده‌ی گوگل (GFS) متولد شد. در واقع Hadoop دارای دو بخش موتور نگاشت-کاهش و سیستم فایل توزیع شده Hadoop یعنی HDFS است. در یک خوشه‌ی کوچک از Hadoop چهار نوع گره با وظایف مختلف وجود دارد. این چهار نوع گره عبارت‌اند از گره‌ی دنبال کننده‌ی کار،[۲] گره‌ی دنبال کننده‌ی وظیفه،[۳] گره‌ی نام[۴] و گره‌ی داده[۱] است. گره دنبال کننده‌ی

---

[1] - MapReduce

[۲] JobTracker

[۳] TaskTracker

[۴]- NameNode

کار، هر وظیفه را به گره مسئول آن وظیفه در خوشه‌ی توزیعی نگاشت می‌دهد. گره دنبال کننده‌ی وظیفه، وظایف را از گره‌ی دنبال کننده‌ی کار می‌پذیرد و آن وظیفه را انجام و سپس نتایج را بر می‌گرداند. گره نام، سیستم فایل، ساختار پوشه‌ها و مکان بلاکهای داده را نگهداری می‌کند. در نهایت، گره‌ی داده، داده‌های HDFS را ذخیره و به گره نام متصل می‌شود[Had].

۱-۳-۱-۲- موتور نگاشت-کاهش

موتور نگاشت-کاهش Hadoop، موظف به اجرای برنامه‌ی نگاشت-کاهش کاربر است. این عملیات شامل تخصیص زیر وظایف‌ها به گره‌های دنبال کننده‌ی وظایف و سپس بازبینی فرایند انجام این عملیات است. یک وظیفه می‌تواند یک نگاشت و یا کاهش باشد[Had].

HDFS -۲-۱-۳-۲

سیستم فایل توزیع شده Hadoop برای اجرا روی هزاران‌هزار ماشین طراحی شده‌است. این سیستم تا سطح بالایی تحمل خطا دارد. HDFS، یک معماری رئیس/پیرو را پیاده‌سازی کرده‌است. لایه‌ی HDFS شامل یک گره رئیس است که به آن گره‌ی نام گویند. هر فایل به بلاکهایی شکسته و سپس جهت ذخیره توسط گره‌های داده، در طول خوشه توزیع می‌شود. گره‌های داده مسئول اجرای تمام درخواست‌های خواندن و نوشتن از سوی گره‌ی رئیس هستند. گره‌های داده، بلاکهای HDFS را با عملیات ایجاد، حذف و تکثیر مدیریت می‌کنند.

HBase -۲-۱-۴

HBase یک روش ذخیره‌سازی داده بر پایه‌ی پایگاه‌داده‌های کلید-مقدار NOSQL است. این سیستم در واقع پایگاه داده Hadoop است. همانطور که اشاره شد، Hadoop، یک مخزن مقیاس‌پذیر و

---

[۱] DataNode

توزیع‌شده از داده‌های بزرگ است که امکان دسترسی ترتیبی را به کاربران می‌دهد. HBase در مواقعی مورد استفاده قرار می‌گیرد که نیاز به دسترسی تصادفی و بی درنگ به داده‌های حجیم باشد. هدف از پروژه HBase، ایجاد جداول بزرگ با بینهایت سطر و ستون روی خوشه‌ای از سخت افزارهای معمولی است. این سیستم یک مخزن کد باز، توزیعی، نسخه‌بندی[۱]شده و مبتنی بر ستون[۲]است که بعد از جداول بزرگ[۳]گوگل مدل‌سازی شده‌است. HBase نرمال‌سازی جداول و پشتیبانی کامل از ACID را با عملکرد بالا معامله کرده‌است. بنابراین جداول غیر نرمال هستند و خصوصیات منعطف‌تر از ACID روی جداول وجود دارد[Med11][HBase].

۱-۴-۱-۲- مدل داده HBase

مهمترین واحد منطقی HBase ستون است. چندین ستون یک سطر را تشکیل می‌دهند که توسط کلیدِ آن سطر آدرس‌دهی می‌شوند. یک مجموعه سطر، یک جدول را تشکیل می‌دهند که این جدول براساس سطرها مرتب هستند. ترکیب هر سطر و ستون یک سلول را تشکیل می‌دهد. هر سلول را می‌توان با زمان برچسب‌گذاری نمود و در نتیجه‌ی آن چندین نسخه از یک مقدار را در یک ستون ذخیره کرد.

ستونهای یک جدول به ستونهای خانواده گروه‌بندی می‌شوند. مهمترین نکته این است که ستونهای خانواده هنگام طراحی شِما باید تعریف شوند در حالی که ستونها می‌توانند به صورت پویا هنگام درج داده اضافه شوند. تعداد ستونها و سطرها در پایگاه HBase نامحدود است[Med11]. البته تعداد ستونهای

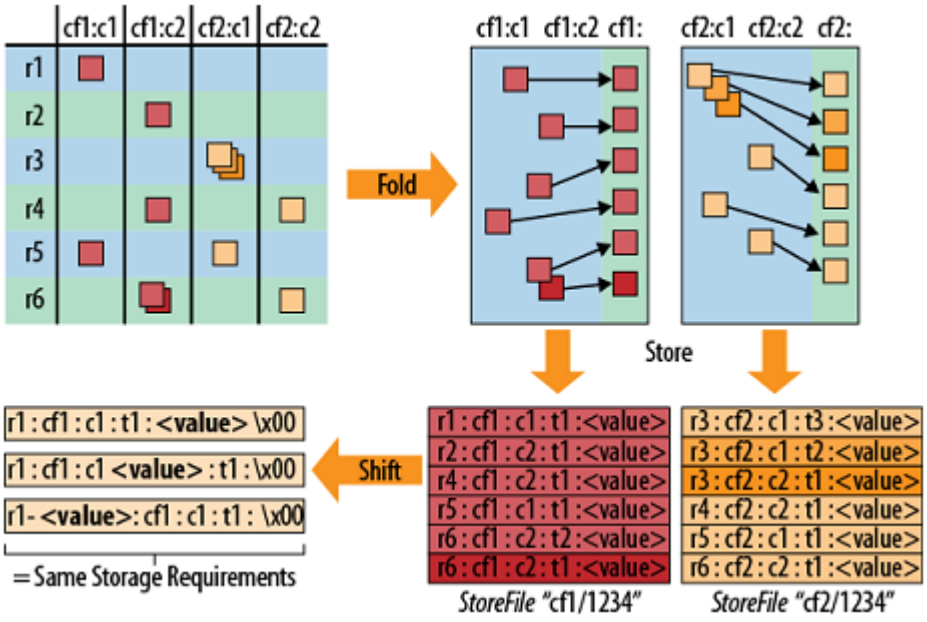
۱- Versioning

۲-Coloumn Oriented

۳-BigTable

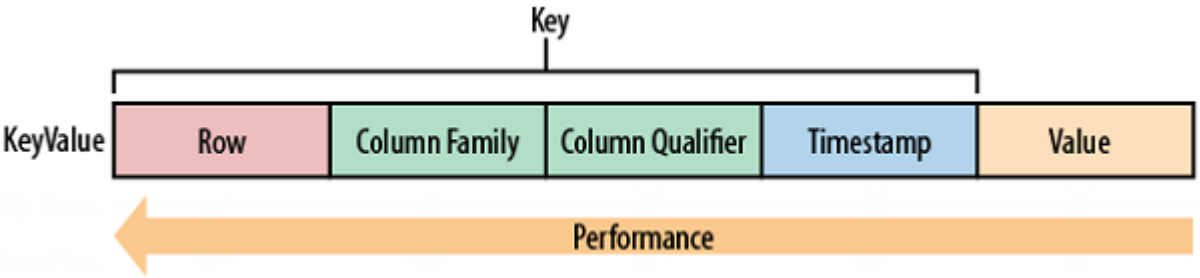
خانواده باید محدود باشد چرا که موجب هزینه ی اضافی i/o می‌شود. تصویر (۱-۲: سمت چپ-بالا) یک مدل منطقی از داده‌ها در HBase را نشان می‌دهد.

همانطور که گفته شده‌است HBase یک پایگاه‌داده‌ی کلید-مقدار است. در مدل فیزیکیِ داده‌ها در HBase چیزی جز کلید-مقدار وجود ندارد. در این قسمت نشان می‌دهیم که کلیدسطر و ستون‌ها چگونه در قالب کلید-مقدار ذخیره می‌شوند. همانطور که در شکل(۲-۱: سمت راست-بالا) مشاهده می‌شود نمای منطقی داده‌ها در قالب نمای فیزیکی نمایش داده‌شده‌است.



شکل ۲-۱: مدل منطقی و فیریکی داده‌ها در HBase [Med11]

در هنگام ذخیره‌سازی (شکل ۲-۱: سمت راست- پایین) ، HBase به ازای هر ستون خانواده یک فایل در نظر می‌گیرد. بنابراین داده‌ها مطابق شکل(۲-۲) قابل دستیابی خواهند بود.



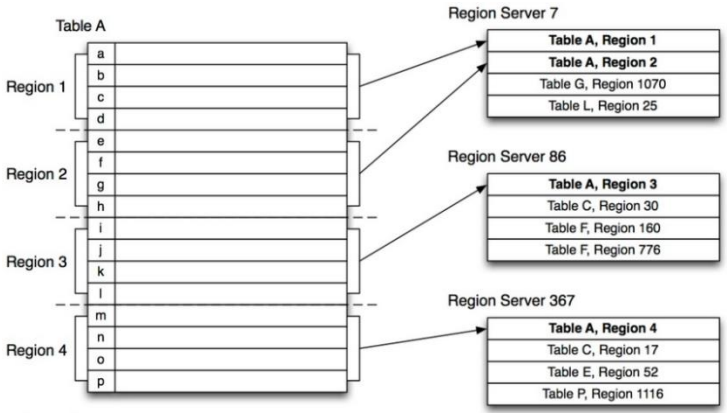
شکل ۲-۲: ساختار کلید و مقدار در مدل قیزیکی ذخیره‌ی داده ها در HBase [Med11]

اما درادامه به توضیح شکل(۱-۲: سمت چپ-پایین) می‌پردازیم. مهمترین چیز در طراحی کلید سطر و ستون‌ها این است که می‌توان داده‌ها را از مقادیر سلول به ستون‌ها و یا حتی کلیدهای سطر قرار داد. با این عمل زوج کلید-مقدار با همان اطلاعات بوده و فقط یک بیت شیفت به چپ دارند. به عنوان مثال اگر مقدار V با کلید سطر K در ستون خانواده CF باشد، بنابراین می‌توان آن مقدار را در سلول ذخیره کرد[K:CF:C:timestamp:V] و یا آن را در کلید سطر همراه با کلید [KV:CF:C:timestamp:0] ذخیره کرد. خصوصیت دومین مدل در این است که با این نوع ذخیره‌سازی، HBase امکان جست‌وجوی پیشوندی را به کاربر می‌دهد. بنابراین کاربر می‌تواند داده‌ها را هم با k و هم kv بازیابی کند. اما همانطور که از شکل(۲-۲) نیز پیداست، مهمترین مزیت این نوع طراحی، افزایش نرخ بازیابی داده‌ها هنگام استفاده از تنها کلید سطر است. با این توضیحات به معرفی یک بحث مهم در طراحی جداول HBase می‌پردازیم. با تمرکز به شکل (۱-۲: سمت چپ-پایین) در HBase دو نوع طراحی از جدول وجود دارد. مدل اول، مدل ستون‌های کم و سطرهای زیاد و مدل دوم مدل ستون‌های زیاد و سطرهای کم است. تفاوت آنها در ذخیره‌سازی، به نحوه‌ی قرار گیری مقدار یک سلول بر می‌گردد که در شکل (۱-۲: سمت چپ-پایین) سه نوع ذخیره‌سازی مطرح شده‌است. هر کاربر بسته به نوع طراحی خود از یکی از این نوع طراحی‌ها استفاده می‌کند. در مدل ستون‌های کم و سطرهای زیاد، امکان جست‌وجوی مبتنی بر پیشوندی فراهم است. اما در این نوع ذخیره‌سازی تعداد سطرها برای یک موجودیت بسیار زیاد می‌شود و بروزرسانی و بازیابی یک موجودیت را دچار پیچیدگی می‌کند. به عنوان مثال برای بازیابی یک موجودیت با یک سری از صفات

خاص، نیاز به عمل الحاق روی این جداول بزرگ است که پیچیدگی پردازش را بالا خواهد برد. اما در روش دوم تمام اطلاعات یک موجودیت به صورت یکجا در دسترس خواهد بود. تنها اگر تعداد ستون‌ها خیلی زیاد باشد، ممکن است به دلیل حضور یک موجودیت با ستون‌های با حجم زیاد، مشکل عدم توازن بار پیش‌آید. همچنین ممکن است داده‌ها به نحوی پراکنده باشند که جست‌وجوی یک موجودیت با یک سری از صفات، سرعت پردازش را بالا ببرد.

۲-۴-۱-۲- معماری HBase

هر جدول در HBase به ناحیه‌های افقی تقسیم و به سرورهای ناحیه‌ای داده می‌شود. همان طور که در شکل(۲-۳) نشان داده شده، هر سرور ناحیه‌ای در هر لحظه، تنها یک ناحیه مشخص را دارد و بنابراین سازگاری بالا در HBase به این شکل تامین می‌شود. ناحیه‌ها واحد پایه‌ی دسترس‌پذیری و توزیع در جداول HBase هستند.



شکل ۲-۳: مفهوم ناحیه در جداول

به ازای هر ستون خانواده در HBase یک مخزن[۱] تشکیل می‌شود. هر ناحیه مجموعه‌ای از این مخزن‌ها است. در سرور ناحیه‌ای برای هر مخزن از یک ناحیه‌ی جدول، داده‌ها توسط memStore در

[۱]- Store

حافظه اصلی ذخیره می‌شود. هنگامی‌که یک memStore به حد نصاب برسد، به ازای هر ستون خانواده از یک ناحیه‌ی جدول، داده‌ها در حافظه‌ی دائمی به نام storeFile ذخیره می‌شوند. هر storfile نیز خود به بلاکهایی شکسته می‌شود که از طریق یک شاخص بلاک می‌توان به آن دسترسی یافت.

یک خوشه‌ی کوچک از HBase دارای سه نوع گره رئیس[۱] هماهنگ‌کننده[۲]و گره‌های ناحیه‌ای[۳] است. گره‌ی هماهنگ‌کننده مسئول ایجاد هماهنگی بین اعضای خوشه‌است. گره رئیس مسئول توازن بار روی خوشه‌ی توزیعی است. این گره، جداول را به ناحیه‌هایی تقسیم و به سرورهای ناحیه‌ای انتساب می-دهد. همچنین با استفاده از گره‌ی هماهنگ‌کننده از وضعیت گره‌های ناحیه‌ای باخبر می‌شود. گره‌های ناحیه‌ای هم که اشاره به سرورهای ناحیه‌ای دارد.

چهار عمل اساسی در HBase وجود دارد که شامل درج، حذف، پویش و بازیابی می‌باشد. عملیات درج شامل عملیات بروزرسانی و یا درج یک موجودیت است. اگر کلید درج شده در جدول وجود داشته باشد، مقدار ستون با مقدار جدید جایگزین می‌شود[Med11].

۳-۴-۱-۲- ویژگی‌های پیشرفته‌ی HBase

در این بخش به معرفی ویژگی‌هایی از HBase پرداخته می‌شود که در سیستم پیشنهادی مورد استفاده قرار گرفته‌است. این ویژگی‌ها شامل صافی‌ها[۴] Bloom Filters، مخزن اتصالات[۱]و پشتیبانی از داده‌های پراکنده[۲] می‌باشد[Med11].

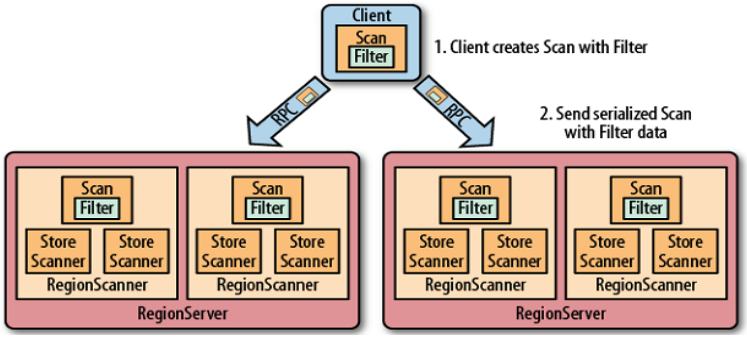
۱- Master

۲- Zookeeper

۳- Region Server

۴- Filters

صافی‌ها– صافی‌ها یکی از مهمترین ویژگی‌ها در HBase هستند. با کمک این ویژگی می‌توان در هنگام پرس‌وجو امکان کنترل بهتر روی داده‌ها داشت. همانطور که از شکل(۴-۲) نیز پیداست، مهمترین مزیت در ارائه این امکان، پیاده‌سازی آنها در سمت سرور است.



شکل ۲-۴ :مفهوم صافی در HBase [Med11]

بنابراین اگر بار داده‌ها به خوبی توزیع شده باشدکل مجموعه توزیع شده مورد استفاده قرار می-گیرد. به عنوان مثال می‌توان کلیدهای جداول را طوری سازماندهی کرد که بتوان از طریق جست‌وجوی پیشوندی آنها را بازیابی نمود. صافی‌ها قابلیت ترکیب با صافی‌های دیگر را دارند و بنابراین می‌توان همزمان چندین صافی روی سطر و ستون و مقدار سلول‌ها ایجاد کرد. البته HBase از صافی‌های سلسله-مراتبی حمایت نمی‌کند.

Bloom Filters– Bloom Filters به عنوان یک ابزذخیره[۳]برای Storfile محسوب می‌شود. این امکان باعث می‌شود که دفعات خواندن سطرها در طول یک پویش جدول کاهش یابد. زمانی که یک Storfile باز می‌شود Bloom Filters مربوطه به حافظه آورده شده و قبل از اینکه Storfile پویش گردد با

---

[۱]- Thread Pool

[۲]- Sparse

[۳]- MetaStor

استفاده از Bloom Filters وجود یک سطر در Storfile مشخص می‌شود. در نتیجه بلاکهایی از Storfileکه مقدار مورد نظر را ندارند مورد جست‌وجو قرار نمی‌گیرند.

مخزن اتصالات- دسترسی به جداول در HBase عملیاتی زمان بر است و حدود چند ثانیه طول می‌کشد. با استفاده از این امکان، برای هر درخواست که از طرف مشتری می‌آید، تنها یک نمونه از جدول ایجاد و سپس مجددا از آن استفاده می‌شود.

پشتیبانی از داده‌های پراکنده- با توجه به اینکه داده‌های مختلف در حال ذخیره‌سازی در جداول هستند ممکن است به ازای یک سری از ستونها، بعضی از سطرها دارای مقدار نباشند. با توجه به مدل فیزیکی HBase برای این مقادیر *NULL* فضایی در دیسک در نظر گرفته نمی‌شود. بنابرانین بر خلاف جداول رابطه‌ای سربار *NULL* در حافظه وجود ندارد.

۲-۲- کارهای مرتبط

در ارزیابی کیفیت و مقیاس‌پذیری یک سیستم شاخص‌گذاری همواره دوعامل تاثیر گذار است. عامل اول مدل داده‌ای است که برای ذخیره و بازیابی داده‌های RDF ارائه می‌شود. به مدل ذخیره داده-های RDF، شِمای شاخص‌گذاری نیز می‌گویند. شِمای شاخص‌گذاری باید به نحوی طراحی شود که هم در ذخیره و هم در بازیابی داده‌ها عملکرد مطلوبی را ارائه دهد. به عنوان مثال مدل داده پیشنهادی باید افزونگی کم داشته و در عین حال داده‌های مرتبط با پرس‌وجو در کمترین زمان ممکن یافت و بازیابی شوند. در بخش‌های بعدی به این ویژگی‌ها مفصل‌تر اشاره خواهد شد. عامل تاثیر گذار دوم سیستمی است که برای مدیریت شِمای شاخص‌گذاری RDF انتخاب می‌شود. با توجه به مدل ذخیره داده‌های RDF، سیستم‌هایی برای مدیریت داده‌ها براساس شِمای انتخابی وجود دارند.

بر این اساس در ادامه ابتدا در بخش (۲-۲-۱) شِماهای ارائه شده در زمینه‌ی شاخص‌گذاری، دسته‌بندی و سپس بر اساس یک سری از معیارهای ارزیابی معرفی شده در بخش (۲-۲-۲)، مورد مقایسه قرار می‌گیرند. در بخش (۲-۲-۳) انواع سیستم‌های مدیریت شاخص‌ها معرفی می‌شوند. سپس معیارهای ارزیابی یک سیستم مدیریت شاخص معرفی و درنهایت در بخش(۲-۲-۴) امکانات سیستم‌ها به طور کلی در یک جدول مقایسه با یکدیگر مقایسه می‌شوند.

۲-۲-۱- انواع شِماهای شاخص‌گذاری RDF

روش‌های مختلفی برای شاخص‌گذاری داده‌های RDF ارائه شده‌است که هر یک در طراحی شِمای شاخص‌گذاری خود، تفسیرهای متفاوتی از مدل داده‌ی RDF ارائه کرده‌اند. این روش‌ها به دو دسته‌ی کلیِ مبتنی بر ساختار گرافی RDF و مبتنی بر ساختار سه/چهارگانه RDF تقسیم می‌شوند. شِماهایی که در گروه شِماهای مبتنی بر ساختار گرافی‌شکل RDF قرار دارند[Mah12] [Lev09] [Wil03]، ابتداروی ساختار گرافی RDFها تحلیل‌هایی انجام می‌دهند تا بتوانند یک سری الگو کشف و سپس با استفاده از الگوها عمل شاخص‌گذاری را انجام دهند. اما شِماهایی که مبتنی بر ساختار سه‌گانه اسنادRDF هستند[Owe08][Neu10]، سربار تحلیل اولیه را حذف می‌کنند. در واقع عمل شاخص‌گذاری در این نوع از شِماها به نحوی انجام می‌گیرد که بتوان ساختار داده‌ها را با انجام یک سری از الحاق‌ها استخراج نمود. در ادامه به شرح کارهای پیشین بر اساس این دو دسته پرداخته می‌شود.

۲-۲-۱-۱- شِمای شاخص‌گذاری مبتنی بر ساختار گرافی RDF

روش‌های شاخص‌گذاری موجود که این شِماها را طراحی نموده‌اند، شامل شاخص‌گذاری مبتنی بر گراف[1][Agg10][Yan04]، شاخص‌گذاری مبتنی بر ساختار[Mch97]، شاخص‌گذاری مبتنی بر

---

[1]-Graph Indexing

predicateهای مشترک[Wil03]، شاخص‌گذاری مبتنی بر بخش‌بندی بر اساس ساختار[Tha12] ، و شاخص‌گذاری مبتنی بر گره برچسب‌گذاری شده[Del10] هستند. در ادامه دو نمونه از مرتبط‌ترین روش-ها با سیستم پیشنهادی معرفی می‌شود.

شاخص‌گذاری مبتنی بر predicateهای مشترک

این نوع شاخص‌گذاری، ساختار گرافی RDFها را برای یک سری تحلیل‌ها در نظر می‌گیرد و سپس در قالب عناصرسه/چهارگانهRDF یعنی (s,p,o) آنها را در جداول رابطه‌ای ذخیره می‌کند. برای اینکه این شِما بتواند با این نحوه‌ی ذخیره‌سازی جوابگوی پرس‌وجوهای گرافی شکل باشد، یا Prdicateهای مشترک را برای subjectها خوشه‌بندی می‌کند[Lev09][Wil03][Wil06] و یا یک‌جا ستون‌های جدول را برای کل صفات در نظر گرفته و برای هر subject یا به عبارتی موجودیت، یک ردیف را اختصاص می‌دهد. این روش بیشتر برای پایگاه‌داده‌هایی پیشنهاد می‌شود که سربار *NULL* نداشته‌باشد[JADID].

می‌توان گفت که از درون یک مخزن داده می‌توان حجم زیادی از نظم را یافت. به عنوان مثال در مخزن داده کاربران، شماره پرسنلی، نام و... برای همه کارمندان ممکن است وجود داشته باشد. بنابراین راهکاری که به ذهن می‌رسد این است که این صفات مشترک در کنار هم وجود داشته و براساس آن subjectها دسته بندی شوند. جدول ویژگی jena به عنوان یک جدول رابطه‌ای در نظر گرفته می‌شود[Wil03] که هر سطر آن متعلق به یک یا چند عبارت RDF است. Jena برای ذخیره سه‌گانه‌هایی مانند شکل (۵-۲) از جداول مختلف استفاده می‌کند.

---

[۱]-Structure Indexing

[۲]-commone predicate

[۳] Structure oriented Partitioning

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

شکل۲-۵: سه‌گانه‌های نمونه برای ذخیره در جداول ویژگی[Aba07]

جدول ویژگی‌های خوشه‌بندی شده[۱] این جدول در شکل(۶-۲) نشان داده شده‌است. این شِما برای صفات خوشه‌بندی شده‌ای است که حداکثر یک مقدار دارند. ستون subject به عنوان کلید جدول عمل می‌کند. هر جدولِ ویژگی ممکن است یک مقدار از object را ذخیره نماید و یا *NULL* باشد.

جدول سه‌گانه‌ها- این جداول برای ذخیره‌ی سه‌گانه‌هایی است که در هیچ یک از جداول دیگر جای نگیرند.

جدول ویژگی چند تایی[۲] این جدول، برای صفاتی است که در مقدار object دارای چند مقدار باشند. مشکل این نوع ذخیره‌سازی این است که سیستم باید بداند کدام صفت چند مقداری است.

---

[۱]- Clustred Property

[۲]- Multiple property Table

Property Table

Subj.	Type	Title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL
ID5	CDType	"GHI"	"1995"
ID6	BookType	NULL	"2004"

Left-Over Triples

Subj.	Prop.	Obj.
ID1	author	"Fox, Joe"
ID2	artist	"Orr, Tim"
ID2	language	"French"
ID3	language	"English"

شکل ۲-۶: جدول ویژگی و جدول سه‌گانه، [Aba07]

بعد از ارائه این شِمای ذخیره‌سازی، روش دیگری جهت رفع عیوب آن ارائه شد[Lev09]. این روش راهکاری را برای رفع مشکل خوشه‌بندی و حضور مقادیر *NULL* ارائه داده‌است. در این راهکار هدف، خوشه‌بندی موجودیت‌ها و قرار دادن آنها در جداولی مشترک است. با توجه به حضور مقادیر *NULL* و هزینه‌ی الحاق در جداول رابطه‌ای، هدف از الگوریتم ارائه شده، کاهش مقادیر *NULL* و هزینه الحاق است. زیرا هرچه هزینه‌ی الحاق پایین باشد و ستون‌ها بیشتر شوند مقادیر *NULL* افزایش می‌یابد.

در این روش خوشه‌ها در دوفاز ایجاد می‌شوند. در فاز اول دسته صفات و یا Predicateهایی که برای هر موجودیت رخ داده‌است به همراه تعداد تکرار آنها استخراج می‌شوند. این خوشه‌ها هر یک دارای دو مقدار درصد حمایت[۱]در درون مجموعه‌داده و درصد *NULL* هستند. سپس بر اساس درصد حمایت، به ترتیب صعودی، خوشه‌ها مرتب می‌شوند. خوشه‌هایی که درصد *NULL* پایین و حمایت بالا دارند و نیز اعضای درون این خوشه‌ها(صفات) که در جای دیگر ظاهر نشده‌اند تشکیل یک جدول را می‌دهند. در فاز دوم، مجددا خوشه‌ها براساس حمایت آنها به صورت صعودی مورد پردازش قرار می‌گیرند. اگر خوشه، درصد *NULL* بالایی داشت، آنگاه از صفات عضو آن خوشه، صفتی که بیشترین *NULL* را تولید کرده‌است

[۱]- Support Percent

حذف و اگر آن صفت در دیگر خوشه‌های باقی مانده وجود نداشت، آن صفت یک جدول ایجاد می‌کند که نام آن جدول، نام صفت مورد نظر و subject و object ستون جدول است. این کار آنقدر انجام می‌گیرد که خوشه‌ی مورد نظر به میزان *NULL* قابل قبول برسد. سپس برای آن جدولی ایجاد و سپس به علت عدم وجود هم‌پوشانی تمامی‌صفات درون این خوشه از درون دیگر خوشه‌های باقیمانده حذف می‌شود.

این نوع شِما به طور مشابه، در [JADID] و روی سیستم HBase ارائه شده‌است. در واقع در شِمای ارائه شده، موجودیت‌ها به عنوان کلیدِ سطر و صفات، ستون جدول هستند. در ضمن هیچگونه عملیات خوشه‌بندی انجام نشده و موجودیت‌ها پس از استخراج از سند در HBase ذخیره می‌شوند.

ارزیابی کلی- موجودیت، به داده‌هایی گفته می‌شود که به توصیف یک سری از اقلام اطلاعاتی می‌پردازد. این شیوه از شاخص‌گذاری می‌تواند به بازیابی یک موجودیت کمک موثری کند؛ چرا که هر subject با objectهایی که توصیف کرده در یک جدول ذخیره شده‌است.

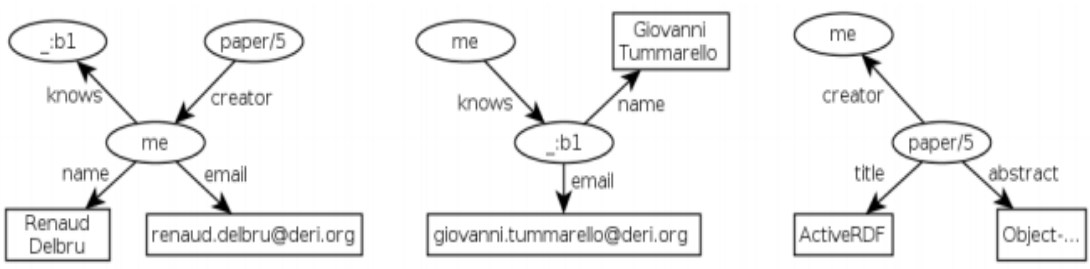
در طراحی این نوع شِما روی جداول رابطه‌ای چالش‌هایی وجود دارد[Aba07]. ابتدا برای خوشه‌بندی داده‌ها زمان زیادی گذاشته می‌شود به خصوص اینکه اگر خوشه‌بندی قرار است بهینه هم باشد؛ یعنی مقادیر *NULL* کمتری در جداول ایجاد کند و درعین حال حجم جداول نیز زیاد نشود. همچنین پرس‌وجوهایی که predicate آنها متغیر باشد و ذکر نشده باشد بازهم همین مشکل را به وجود می‌آورد. مشکل دیگر تغییر مرتب شِما برای خوشه‌بندی predicateها است و بسته به خوشه‌ها، جداول مختلف با سطروستون‌های مختلف ایجاد می‌شود که مطلوب نیست. وجود مقادیر *NULL* مشکل بعدی این ساختار است. در این ساختار به دلیل بهینه‌سازی، subjectهایی که در خصوصیت کمی با یکدیگر فرق کنند در یک جدول قرار داده می‌شوند. بنابراین برای subjectهایی که predicate خاصی برای آن تعریف نشده، مقدار *NULL* گذاشته می‌شود که هنگام ذخیره سازی داده‌ها با مقیاس بالا سربار مقادیر*NULL* افزایش میابد. مشکل دیگر این شِما عدم کنترل منعطف predicateهای چند مقداره‌است. با توجه به اینکه

ممکن است predicateهایی همچون ایمیل دارای چند مقدار باشند این نوع سه‌گانه به دلیل کلید اصلی بودن Subject در جدول ویژگی حمایت نمی‌شود. بنابراین objectها باید در جدول دیگر (جدول ویژگی چندتایی) ذخیره شود که بسیار غیر منعطف است.

در طراحی این نوع شِما روی HBase نیز معایبی وجود دارد. این نوع شِما که نام آن را شِمای تک جدولی می‌نامیم، سعی در رفع ایرادات گفته شده در جداول رابطه‌ای دارد. به عنوان مثال، مشکل صفت-های چند مقداری با ویژگی نسخه‌بندی ستون‌های HBase حل شده‌است. در این نوع شِما خوشه‌بندی صورت نمی‌گیرد و داده‌ها در یک جدول بزرگ ذخیره می‌شوند. در طراحی این نوع شِما، از روش طراحی جدول در HBase که مبتنی بر ستونهای زیاد و سطر کم است استفاده شده‌است. اما همانطور که در بخش(۱-۲) هم مطرح شد، با توجه به اینکه موجودیت‌ها در تعداد صفات مورد پشتیبانی با یکدیگر متفاوت هستند، هنگام جست‌وجو در جداول نرخ بازیابی یک موجودیت با صفات خاص پایین می-آید[JADID].

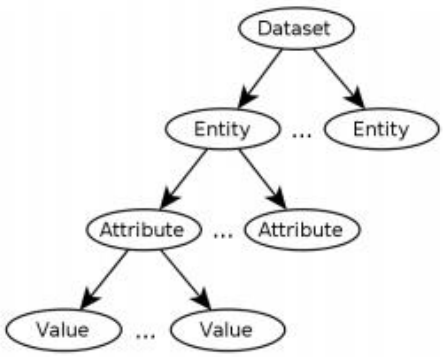
شاخص‌گذاری مبتنی بر گره برچسب گذاری شده

این شاخص‌گذاری توسط سیستم بازیابی اطلاعات SIREn ارائه شد و در موتورجستجوی Sindice[Del10] به کار گرفته شد. در این گونه ذخیره‌سازی فرض بر این است که واحد اصلی اطلاعاتی که باید مورد جستجو قرار بگیرد، موجودیت است[Del10]. موجودیت به اقلامی‌اشاره دارد که به توصیف یک سری اطلاعات می‌پردازند. به عنوان مثال این اقلام می‌توانند یک شخص، مکان یا محصول را توصیف کنند. در یک گراف داده، ساده‌ترین فرم یک گره موجودیت یک گراف ستاره‌ای است. در شکل (۷-۲) نمایی از این توصیفات را در قالب گراف ستاره‌ای می‌بینیم.



شکل ۲-۷: نمونه‌ای از گرافRDF [Del10]

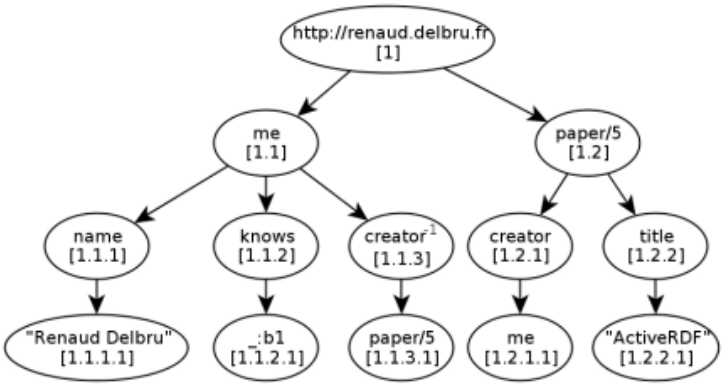
در این روش، از مدل برچسب گذاری گره‌های درخت جهت یافتن رابطه‌ی بین منابع داده، موجودیت‌ها، ویژگی و مقدار استفاده می‌شود. با توجه به شکل(۷-۲) یک درخت چهار نوع گره دارد. برای یک گراف RDF این درخت را می‌توان همانند شکل (۸-۲) رسم نمود.



شکل ۲-۸: ارئه مفهومی‌از مدل درختی مبتنی بر گره‌های بچسب گذاری شده[Del10]

با توجه به شکل (۹-۲) هر ترم، بعد از قرار گرفتن در درخت و انجام عمل برچسب گذاری با استفاده از کدگذاری deway، موقعیتش مشخص شده و لیست‌های معکوس آن ساخته و در فایل قرار می‌گیرد. سپس هر ترم در دیکشنری به نحوی ذخیره می‌شود که به لیست‌های خود در فایل اشاره می‌کند. کدگذاری deway درواقع گره‌های یک درخت را به نحوی شماره‌گذاری می‌کند که گره‌های هر شاخه‌ی درخت دارای پیشوند مشترکی از اجداد خود باشند. بنابراین منظور از لیست‌های معکوس، لیست

اعداد به دست آمده از کدگذاری اشاره شده برای هرگره‌است. برخلاف آنچه که در شکل نشان داده‌شده‌است، به جای استفاده از اعداد متوالی، از شناسه ترم‌ها استفاده می‌شود.



شکل ۲-۹: مدل درختی داده‌ها با استفاده از کد گذاری deway [Del10]

نحوه‌ی دسترسی به داده‌ها نیز به این شکل است که برای هرترم، لیست معکوس آن بازیابی می‌شود. سپس اگر پیشوند یکسانی از اعداد صحیح در دو لیست وجود داشت آنگاه آن دو از یک شاخه درخت می‌باشند و این یعنی اینکه مربوط به یک موجودیت هستند.

ارزیابی کلی- ارائه مهمترین نوع جستجو یعنی جستجوی مبتنی بر موجودیت بدون هزینه زیاد، می‌تواند به عنوان یک نکته خوب برای این سیستم محسوب شود. چرا که دسترسی به شاخص‌ها از طریق اشاره‌گر، باعث شده است که هزینه‌ی جستجو در درخت دودویی ویا Btree را که شِماهای دیگر دارند، این شِما نداشته‌باشد. چالش این نوع شِما عدم حمایت آن از پرس‌وجوهای پیچیده‌است. البته با توجه به اینکه هر ترم مسیری از ریشه یعنی موجودیت تا مکان خود را در لیست نگه‌داری می‌کند برای یافتن یک عبارت سه/چهارگانه نیاز به الحاق، البته با هزینه کم است.

۲-۱-۲-۲- شمای شاخص‌گذاری مبتنی بر ساختار سه /چهارگانه RDF

دومین دسته از روش‌های شاخص‌گذاری، این نوع شِماها هستند. روش‌های شاخص‌گذاری موجود که به این شکل شِماهای خود را طراحی نموده‌اند، شامل شاخص‌گذاری یکپارچه[۱][Neu10]، شاخص‌گذاری مبتنی بر بخش‌بندی بر اساس predicate و به اصطلاح بخش‌بندی عمودی[۲][Aba07]، شاخص‌گذاری مبتنی بر بخش‌بندی بر اساس تمامی‌عناصر سه‌گانه[Wei08]، شاخص‌گذاری ترکیبی[۳][Kha12]، شاخص‌گذاری مبتنی بر فیلد[۴][Luc12] است. در ادامه به شرح سه مورد از مرتبط‌ترین روش‌ها با روش پیشنهادی معرفی می‌شود.

شاخص‌گذاری یکپارچه

در این شِما یک جدول بزرگ سه/چهارستونه برای ذخیره‌سازی سه‌گانه‌ها مورد استفاده قرار می‌گیرد. ازجمله سیستم‌هایی که از این شِما استفاده می‌کنند عبارت‌اند از: [Neu10] ، [Owe08] , [SDB]، [Ses] ، [Har07]، [Pap12] و موتورجستجوی SWSE [Hog11] است.

برای دسترسی با سرعت بیشتر به عناصر سه/چهارگانه، باید الگوهای مختلف از ترتیب این سه/چهارگانه‌ها را در نظر گفت. با توجه به اینکه ۱۶ الگوی دسترسی به عناصر چهارگانه (?,?,?,?) وجود دارد، سیستم‌ها تنها الگوهای دسترسی رایج را شاخص‌گذاری می‌کنند. در سال ۲۰۰۵روشی ارائه شد، که تنها با شش شاخص از چهارگانه‌ها توانست همه این الگوهای دسترسی را حمایت کند[Har05].

---

۱ Monolithic Indexing Schema

۲ - Vertical Partitioning

4 0 – Hybrid

4 2- Field Index

در ذخیره‌سازی چهارگانه‌ها، ۱۶ نوع الگوی دسترسی وجود دارد. با بررسی روی الگوی دسترسی می‌توان وجود یک سری از همپوشانی‌ها را یافت و آنها را به صورت شکل (۲-۱۰) دسته بندی نمود.

Index	Query patterns
SPOC	(?, ?, ?, ?), (s, ?, ?, ?), (s, p, ?, ?), (s, p, o, ?), (s, p, o, c)
CP	(?, ?, ?, c), (?, p, ?, c)
OCS	(?, ?, o, ?), (?, ?, o, c), (s, ?, o, c)
POC	(?, p, ?, ?), (?, p, o, ?), (?, p, o, c)
CSP	(s, ?, ?, c), (s, p, ?, c)
OS	(s, ?, o, ?)

شکل ۲-۱۰: پوشش الگوهای دسترسی توسط شش شاخص، [Har05]

جهت افزایش مقیاس‌پذیری، این نوع شِما روی سیستم متشکل از HBase و چهارچوب Mapreduce نیز پیاده سازی شده‌است[Pap12]. در واقع سیستم‌های متمرکز نسبت به سیستم‌های توزیع شده دربرابر پرس‌وجوهایی که گزینش‌پذیری بالا دارند (تعداد متغیرها در پرس‌وجوهای آنها کم است) بهتر عمل می‌کنند. بنابراین این سیستم از یک راه‌کار تطبیقی برای الحاق‌ها استفاده می‌کند تا از قابلیت هر دو نوع ذخیره‌سازی بهره مند شود. در شِمای ذخیره‌سازی از سه شاخص (سه جدولspo,pos,osp) استفاده شده‌است. به عنوان مثال در جدول spo کلید سطر برابر sp و ستون o در نظر گرفته می‌شود. همچنین این شِما توسط Jena [Kha12] روی جداول مبتنی بر ستون HBase پیاده سازی شده‌است. نام این ساختار در jena، ساختار شاخص‌گذاری شده است.

ارزیابی کلی- این شِما با طراحی ساده و پوشش الگوهای مختلف دسترسی، عملکرد خوبی را ارائه داده است. تنها چالشی که وجود دارد، وجود جداول مختلف برای پشتیبانی از الگوهای دسترسی است؛ که دو مشکل فضا و بروزرسانی را ایجاد می‌کند. به ازای هر سه‌گانه که وارد می‌شود، باید حداکثر شش جدول

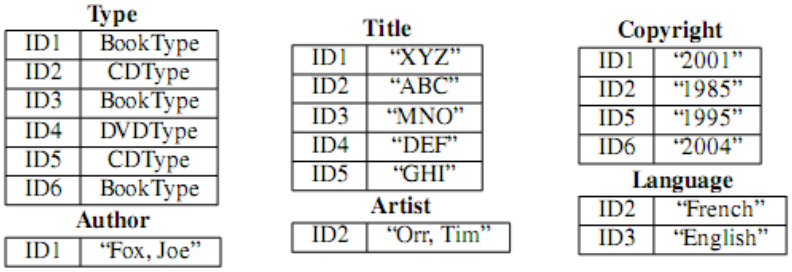
۱- Indexed layout

بروزرسانی شوند[Har05]. برای رفع این دو مشکل RDF3x از فشرده سازی و بروزرسانی تاخیری استفاده می‌کند که توانسته کارایی خوبی را ارائه دهد. در بروزرسانی تاخیری، تغییرات آنی به جداول اعمال نمی شود؛ بلکه در یک فضای دیگر نگهداری و سپس با پر شدن بافر، به داده‌های اصلی اضافه می‌شود. در هرصورت RDF3x یک سیستم متمرکز است و قادر به پاسخ‌گویی بالا اما تا حجم محدود را داراست. حال هرچه مقیاس داده‌ها بزرگ شود، مسئله‌ی الحاق به مسئله‌ای بزرگتر و پرچالش تبدیل می‌شود.

شاخص‌گذاری مبتنی بر بخش بندی بر اساس predicate

ایده‌ی شِمای مبتنی بر بخش بندی اولین بار در [Aba07] ارائه شد. شِمای فیزیکی انتخاب شده خوشه‌بندی subject و object بر اساس predicate است. یعنی به ازای هر predicate جدولی ایجاد می‌شودکه شامل دو ستون برای subject و object است.

این ایده بعد از ارائه‌ی شِمای بخش بندی مبتنی بر predicateهای مشترک Jena2 [Wil03] که در بخش (۲-۲-۱-۱) شرح داده شد، بیان شد. در این شِما، جدول ویژگی در شِمای مبتنی بر predicateهای مشترک به صورت عمودی برش داده شد. همانطور که در شکل(۱۱-۲) نیز پیداست، به ازای هر predicate یک جدول تشکیل می‌شود. این کار تمام مشکلاتی که در شِمای بخش بندی مبتنی بر predicate بود را تقریبا برطرف نمود. مشکلی که رفع نشد، حضور پرس‌وجوهایی است که در آنها predicate متغیر است و در این حالت برای پیداکردن یک subject، سیستم مجبور به الحاق بین تمامی-جداول است. بنابراین در حجم زیادِ داده‌ها دیگر پاسخگو به این چنین پرس‌وجوهایی نیست[Neu10] . چون در این شِما کلید اصلی subject و object است، لذا subjectهای یکسان با objectهای متفاوت می‌توانند ذخیره شوند و به این طریق مشکل صفت‌های چند مقداریِ شِمای بخش بندی مبتنی بر predicateهای مشترک حل می‌شود.



شکل ۲-۱۱-شمای بخش بندی عمودی[Aba07]

این شِما در چهارچوب توزیع شدهٔ Hadoop و به عبارتی با بکارگیری سیستم فایل HDFS پیاده سازی شد[Siv12]. در این نوع پیاده سازی به ازای هر Predicate یک فایل ایجاد می‌شود و سپس برای کوچک کردن فایلها جهت پردازش توزیع شده، مجددا براساس نوع object، فایلها کوچکتر می‌شوند.

ارزیابی کلی- این نوع پیاده سازی توانست اکثرمشکلات جدول ویژگی (چند مقداری و NULL و..) را حذف نماید. در این شِما تنها ویژگی‌هایی که نیاز به خواندن دارند از حافظه خوانده می‌شوندکه موجب کاهش هزینه الحاق‌ها شده است؛ چرا که داده‌های نامرتبط خیلی کم‌ترنسبت به شِمای یکپارچه از حافظه خوانده می‌شوند. چالشی که وجود دارد حضور پرس‌وجوهایی است که در آنها predicate متغیر است که باید برای حل آن تدبیری اندیشید. بنابراین برای یک موجودیت خاص، وارد کردن سه/چهارگانه‌ها نیاز به دسترسی به چند جدول دارد. از سویی هنوز مشکل الحاق برای بازیابی یک موجودیت پابرجاست.

شاخص‌گذاری مبتنی بر بخش بندی بر اساس تمامی‌عناصر سه‌گانه

این شِما توسط HexaStore [Wei08] ارائه شد. آنچه که در این روش به آن پرداخته می‌شود این است که در بخش بندی به سه‌گانه‌ها به دید یکسانی نگاه می‌کند. این یعنی اینکه ایده بخش‌بندی عمودی

---

[۱]-Hadoop filesystem (hdfs):http://hadoop.apache.org/hdfs

علاوه بر صفت روی دو عنصر دیگر نیز اعمال شود. در این پیاده سازی از شش الگوی دسترسی [Har05] پشتیبانی می‌شود.

ارزیابی کلی- کاهش اجتماع‌ها و الحاق‌ها یکی از خصوصیات بارز این روش است. به عنوان مثال تمام سیستم‌های قبلی در یافتن پرس‌وجویی همچون "یافتن تمام Subbjectهایی که با دو object خاص با هر ویژگی رابطه دارند" بسیار پیچیده عمل می‌کنند درحالی که این سیستم به راحتی با دو الحاق ادغامی‌بین subjectهای شاخص OSP به نتیجه می‌رسد. مشکلی که وجود دارد و برای آن نقطه ضعف به شمارمی‌آید فضای زیاد حاصل از جداول است که بروزرسانی را دچار مشکل می‌کند.

۲-۲-۲- معیارهای ارزیابی شِماهای شاخص‌گذاری

در این فصل انواع شِماهای ارائه شده از نظر عملکرد به دو گروه شِماهای مبتنی بر ساختار گرافی RDF و شِماهای مبتنی بر ساختار سه/چهارگانه RDF تقسیم شدند. با بررسی شِماهای مختلفی که وجود دارد می‌توان به وجود یک سری ویژگی‌های کلی مشترک در بین همه این شِماها پی برد که در نظر نگرفتن هریک از این ویژگی‌ها در طراحی شِما می‌تواند، عملکرد آن شِما را تحت تاثیر قرار دهد. اگر قرار باشد شِمای شاخص‌گذاری جدیدی طراحی شود که از دیگر سیستم‌های شاخص‌گذاری بهتر عمل کند باید این ویژگی‌ها را در نظر بگیرد.

این ویژگی‌ها عبارت‌اند از: نوع پرس‌وجوی[۱] مورد پشتیبانی، قدرت الحاق داده‌ها، پشتیبانی از صفت‌های چند مقداری[۳] تعداد شاخص‌های چند گانه[۱] روش دسترسی به شاخص‌ها از حافظه اصلی،

---

[۱]-Query

[۲]-Join

[۳]-Multi Vulue Attribute

اندازه شاخص، ساختار دیکشنری[۲] هزینه بروزرسانی و مقیاس‌پذیری پردازشی است. در ادامه، هر یک از این ویژگی‌ها و نحوه‌ی برخورد روش‌های شاخص‌گذاری با آنها بررسی می‌شود.

نوع پرس‌و جوی مورد پشتیبانی- تفاوتی که باید بین ساختار وب معنایی و وب معمولی بیان کرد حضور پرس‌وجوهایی روی داده‌های ساخت یافته‌است که شامل الحاق‌های بسیار است. پرس‌وجوهایی که روی داده‌های نیمه‌ساخت یافته اعمال می‌شوند، ساختار داده‌ها را نیز در نظر می‌گیرند که در ادامه شرح داده می‌شوند.

- پرس‌و جوهای مبتنی بر مسیر:[۳] این نوع پرس‌وجوها مجموعه‌ای از سه‌گانه‌ها هستندکه به یکدیگر متصلند. به طوری که object یک سه‌گانه subject سه‌گانه دیگر است. به عنوان مثال نام دانشمندانی که در شهری به دنیا آمده‌اند که در آسیا باشد. یک سری از شِماهای شاخص‌گذاری فقط از این نوع پرس‌وجو پشتیبانی می‌کنند[Mat05]. اما در مجموع اکثر سیستم‌ها در کنار پرس‌وجوهای دیگر، از این پرس‌وجو حمایت کرده و روشهایی نیز برای ارائه آن با کمترین هزینه پیشنهاد داده‌اند[Udr07][Aba07][Wei08].

- پرس‌وجوی ستاره ای:[۴] این پرس‌وجو ترکیبی بیشتر از یک مسیر است و یک گره مشترک بین مسیرها وجود دارد. به عنوان مثال نام مردمی را بازیابی کن که هم نویسنده و هم ویرایشگر باشند و مقاله‌ی آنها توسط کسی ارجاع داده شده باشد. در این نوع پرس‌وجو بیشتر نیاز به الحاق‌های -subject

---

۱-Multiple Index

۲-Dictionary

۳Path Queries

۴-Star Queries

subject برای بازیابی موجودیت‌ها به علاوه الحاق‌های subject-object روی مسیرها است. شِماهای مبتنی بر فیلد[Min08] و مبتنی برگره برچسب گذاری شده [Del10] از این دسته پرس‌وجوها حمایت می‌کنند.

- پرس‌وجوی موجودیت[۱]: این نوع پرس‌وجو همانند پرس‌وجوی ستاره‌ای است. در این نوع پرس-وجو الحاق‌های subject-subject بیشتر انجام می‌گیرد. شِمای مبتنی بر گره برچسب گذاری شده [Del10] از این دسته پرس‌وجوها حمایت می‌کنند.

- پرس جوی مبتنی بر گراف[۲]: این نوع پرس‌وجو متشکل از گره‌ها و لبه‌هایی است که به فرم گراف هستند. بر خلاف پرس‌وجوی ستاره‌ای این نوع پرس‌وجو ممکن است دارای چرخه و حلقه باشد. به عنوان مثال: نویسنده x و مشاور y را بازیابی کن به طوری که هر دو در یک سازمان کار کنند و هر دو یکدیگر را بشناسند و y دارای یک شماره تلفن باشد و x کتابی را منتشر کرده باشد. در این نوع پرس‌وجو باید همه نوع الحاق پشتیبانی گردد. شِماهای ترکیبی Jena [Kha12] و نیز شِمای یکپارچه RDF3x [Neu10] نیز از این پرس‌وجو با استفاده از شاخص‌های چندگانه پشتیبانی می‌کنند.

قدرت الحاق داده‌ها- هر گونه پرس‌وجوی انتخابی برای پشتیبانی مشمول الحاق‌هایی روی گراف RDF می‌شود که باید مورد توجه خاص قرارگیرد تا هزینه پرس‌وجو کاهش یابد. سیستم‌های بازیابی داده‌های RDF، داده‌ها را برای هر الگوی سه/چهارگانه بازیابی و سپس در طول پرس‌وجو، عملیات مربوط به الحاق را انجام می‌دهند. بنابراین برای کاهش هزینه الحاق باید بتوان هم هزینه بازیابی داده‌ها (که شامل هزینه جستجو و I/o می‌شود)، و هم هزینه الحاق داده‌ها را کاهش داد. دو راهکار برای قدرت

۱-Entity Queries

۲-Graph shaped Queries

بخشیدن به الحاق‌ها پیش رو است. راهکار اول، عدم بازیابی داده‌های نامربوط نسبت به پرس‌وجو جهت الحاق است؛ و راهکار دوم، کاهش هزینه عملیات الحاق می‌باشد.

شِمایی همچون شِمای مبتنی بر گراف و یا ساختار [Tha12] [Wei08]، به هدف اعمال راهکار اول طراحی شده‌اند. با انتخاب این نوع شِما اولا هزینه جستجو کم می‌شود و ثانیا به خاطر انتخاب داده‌های مبتنی بر ساختارشان، داده‌هایی مرتبط‌تر با پرس‌وجو بازیابی می‌گردند که از حجم داده‌ها کاسته شده و I/O کمتر می‌گردد. همچنین شِمای مبتنی بر بخش بندی براساس predicate [Aba07] با بازیابی تنها predicateهای مورد نیاز هزینه I/o کاهش داده است. دومین راهکار کم کردن هزینه الحاق از طریق بهینه سازی ترتیب الحاق‌ها همچون سیستم RDF3x [Neu10] و انتخاب الحاق‌هایی کم هزینه همچون الحاق ادغامی‌است. الحاق‌های ادغامی با پوشش الگوهای دسترسی به سه‌گانه‌ها امکان پذیر است چراکه داده‌ها مرتب هستند. الحاق ادغامی یکی از کم هزینه ترین الحاق‌هایی است که هزینه زمانی خطی دارد.

پشتیبانی از صفت‌های چند مقداری- گاهی اوقات یک سری از صفات دارای چند مقدارند مانند ایمیل که شِمای ارائه شده باید بتواند از این گونه صفات پشتیبانی کند. اگر پشتیبانی انجام نشود، جوابهایی برای پرس‌وجو باز گردانده می‌شود که یا کامل نیستند و یا نیاز به عملیات پیچیده‌تر دارند و این ضعف بزرگی به حساب می‌آید. شاخص‌گذاری مبتنی بر فیلد [Min08] و شاخص‌گذاری مبتنی بر predicateهای مشترک Jena این ضعف بزرگ را دارند.

تعداد شاخص‌های چند گانه- برای پوشش الگوهای دسترسی به چهارگانه‌ها از انواع شاخص‌های چندگانه‌استفاده می‌شود. این شاخص‌ها هزینه الحاق را بسیار کاهش می‌دهند به این دلیل که داده‌ها مرتب هستند و الحاق ادغامی با کمترین هزینه روی داده‌ها صورت می‌گیرد. اما باید دقت داشت هرچه تعداد این شاخص‌ها زیاد باشد اولا فضای سربار زیادی وجود دارد و ثانیا هزینه بروزرسانی افزایش می‌یابد[Neu10].

روش دسترسی به شاخص‌ها از طریق حافظه اصلی- به روشی اشاره دارد که سیستم‌ها برای دستیابی به شاخص‌هایش به کار می‌گیرد. این روشها چهار نوع هستند که به ترتیب ذکر می‌شوند.

الف) دسترسی مستقیم به داده مورد نظر از طریق اشاره گر به فایل: از این نوع دسترسی Sindice استفاده کرد و توانست هزینه‌ی جستجو در درخت را با ارائه شِمای شاخص‌گذاری مبتنی بر گره برچسب گذاری شده، حذف کند[Del10].

ب) دسترسی مستقیم به بلاک داده مورد نظر از طریق شاخص پراکنده: از این نوع دسترسی سیستم Yars2 [Har07] در هسته موتور جستجو SWSE [Hog12] استفاده کرده‌است. هدف از ارائه این نوع دسترسی این بود که درخت Btree در حجم زیاد داده مقیاس‌پذیر نیست[Har07].

ج) دسترسی از طریق پیمایش درخت Btree: این نوع دسترسی رایج تر از همه دسترسی‌ها است. اکثر سیستم‌هایی که شاخص چندگانه را مورد پشتیبانی قرار می‌دهند، از این نوع ساختار استفاده می‌کنند[Har05].

اندازه شاخص- اشاره به حجم شاخص مورد استفاده دارد که سربار آن یک نقطه ضعف برای شاخص حساب می‌شود. به کارگیری الگوریتم فشرده‌سازی مناسب نه تنها باعث صرفه جویی در فضای دیسک می‌شود، بلکه بازده I/O را بالا برده و در نتیجه بروزرسانی و بازده پرس‌وجو نیز افزایش می‌یابد. RDF3x [Neu10] و Sindice [Del10] از الگوریتم‌های فشرده سازی به نحوی موثر استفاده می‌کند.

ساختار دیکشنری- مبحثی که به طور مشترک در بین این سیتم‌ها مورد بررسی قرار می‌گیرد، بحث استفاده از دیکشنری تبدیل ترم به شناسه است. جهت بالابردن کارایی سیستم‌ها به هر یک از عناصر سه‌گانه‌ی RDF عددی را نسبت می‌دهند تا به پرس‌وجو و الحاق سرعت بیشتری ببخشند. اما استفاده از دیکشنری خود نیاز به الحاق نهایی برای یافتن رشته عناصر است که هزینه در بر دارد. در این

بحث چالشی که وجود دارد، تبدیل ترم به شناسه و بالعکس است. هدف، دوری از سربار این تبدیلات به خصوص تبدیل شناسه به ترم است[Kha12][Owe08]. روشهای ارائه شده برای تخصیص عدد به عناصر جهت ساخت دیکشنری عبارت‌اند از: استفاده از رشته hash، تخصیص دادن اعداد صحیح به ترتیب ورود سه‌گانه‌ها [Bro03]، تخصیص دادن اعداد صحیح متوالی به ازای همسایگی آنها RDF3x [Neu11]، و تخصیص شناسه‌ای که آدرس دقیق محل قرار گرفتن ترم را نشان می‌دهد[Owe08]. در این بین نیز ممکن است، عناصر به شناسه تبدیل نشوند [Kha12] تا هزینه الحاق نهایی با دیکشنری حذف گردد.

هزینه بروزرسانی- مجموع هزینه‌ی جستجوی رشته‌ها در دیکشنری به علاوه هزینه جستجوی رشته در جدول برای درج یا تغییر است. گاهی به دلیل افزونگی زیاد داده‌ها این هزینه افزایش می‌یابد که راهکارهایی برای آن باید اندیشیده شود. به عنوان مثال در شِمای بخش‌بندی مبتنی بر ساختار، هنوز روشی برای بروزرسانی داده‌ها ارائه نشده‌است. تنها سیستمی‌که به این موضوع به صورت اصولی پرداخته، RDF3x [Neu10] است که برای شش شاخص شش گانه خود به طور موثر استفاده می‌کند.

مقیاس‌پذیری پردازشی- منظور از این بحث این است که روش ذخیره‌سازی شاخص‌ها باید با افزایش حجم داده‌ها قابلیت پردازشی خود را از دست ندهد. به عنوان مثال در بعضی از سیستم‌ها افزایش حجم باعث شده‌است که سیستم قادر به ذخیره‌سازی نبوده و یا در پاسخگویی به پرس‌وجوها کند عمل نماید.

۱-۲-۲-۲- جدول ارزیابی روش‌های شاخص‌گذاری

در بخش بعد به ارزیابی شِماهای شاخص‌گذاری در چند چالش مطرح شده در بالا، پرداخته می‌شود. همانطور که در جدول (۱-۲) و (۲-۲) مشاهده می‌کنید، علامت خط تیره نشان‌دهنده‌ی عدم وجود امکان مورد نظر در شِمای ارائه شده‌است. فیلدهای جدول از نظر مفهومی روشن است.

در این ارزیابی تنها ویژگی‌هایی برای شِماها مورد ارزیابی قرار گرفته‌اند که می‌توان به طور کلی در مورد شِمای ارائه شده اظهار نظر کرد؛ و از بررسی عواملی همچون اندازه شاخص و مقیاس‌پذیری خودداری شده‌است به این دلیل که به محیط آزمایش و سیستم مورد استفاده بستگی دارد.

جدول ۲-۱: مقایسه و ارزیابی جزیی شِماهای شاخص‌گذاری

شِما	نوع پرس‌و-جوی مورد حمایت	قدرت الحاق‌ها	پشتیبانی از صفت‌های چندمقداری	شاخص چند گانه	روش دسترسی	هزینه بروزرسانی (درج)
مبتنی بر گراف [Agg10]	مبتنی بر گراف	عدم بازیابی داده‌های نامربوط	وجود دارد	_	شاخص معکوس	زمان تحلیل ساختار گراف+ زمان شاخص‌گذاری
مبتنی بر ساختار [Mac97]	مبتنی بر گراف	عدم بازیابی داده‌های نامربوط	وجود دارد	_	نامشخص	زمان تحلیل ساختارگراف و ساخت راهنمای داده
یکپارچه [Neu10]	گراف	الحاق ادغامی	وجود دارد	معمولا شاخص رایج	Btree	زمان جستجو در درخت +زمان توازن مجدد درخت
مبتنی بر جدول ویژگی [Aba07]	موجودیت	کاهش الحاق subject-subject	غیر منطعف	روی subject و predicateهای مورد نیاز	Btree	جستجو برای سه‌گانه. + زمان درج
فیلد [Min08]	ستاره ای	حذف الحاق برای موجودیت‌ها	خیر	-	مستقیم از طریق دیکشنری	زمان جستجو در دیکشنری
تک جدولی[Jadid]	مبتنی بر گراف	حذف الحاق برای موجودیت‌ها	بله	-	Btree HBase	هزینه درج داده در HBase

جدول ۲-۲: مقایسه و ارزیابی جزیی شِماهای شاخص‌گذاری (ادامه)

شِما/ویژگی	نوع پرس‌وجوی مورد حمایت	قدرت الحاق‌ها	پشتیبانی از صفت‌های چندمقداری	شاخص چند گانه	روش دسترسی	هزینه بروزرسانی (درج)
بخش بندی عمودی روی predicate [Nue10]، [Kha12]	پرس‌و جوهای با predicate نامتغیر	عدم بازیابی نامربوط+ الحاق ادغامی	وجود دارد	PSO POS	Btree	زمان جستجو دردرخت +زمان توازن مجدد درخت
بخش بندی مبتنی بر ساختار [Tha10]	مبتنی بر گراف	عدم بازیابی داده‌های نامربوط	وجود دارد	PEOS PESO POS PSO	شاخص پراکنده	چالش
بخش بندی مبتنی بر تمامی‌عناصر [Wei08]	مبتنی بر گراف	عدم بازیابی داده‌های نامربوط+ الحاق ادغامی	وجود دارد	شاخص شش‌گانه	نامشخص	زمان جستجو در درخت + زمان جستجو روی جدول
‘گره برچسب گذاری شده [Del10]	ستاره ای	کاهش الحاق در موجودیت +سربار الحاق در چهارگانه	وجود دارد	-	مستقیم از طریق دیکشنری	زمان جستجو در دیکشنری
ترکیبی [Kha12]	گراف	عدم بازیابی داده‌های نامربوط+ الحاق ادغامی	وجود دارد	POS PSO SPO OSP	Btree	زمان جستجو در درخت

۳-۲-۲- انواع سیستم‌های ذخیره و مدیریت شاخص‌ها

نوع سیستم ذخیره‌سازی می‌تواند به عنوان امری مهم در تعیین عملکرد یک شاخص تلقی گردد. جهت شاخص‌گذاری RDFها باید نوع سیستمی‌که شاخص RDF درون آن ذخیره گردد را تعیین کرد. انتخاب هر یک از این سیستم‌ها می‌تواند در نتیجه نهایی گرفته شده از RDFها تاثیر مهمی‌داشته باشد، به این دلیل که امکاناتی که این سیستم‌ها ارائه می‌دهند با یکدیگر متفاوت است.

سیستم‌های ذخیره و مدیریت داده‌های RDF معمولا به چند دسته تقسیم می‌شوند. این سیستم‌ها عبارت‌انداز: پایگاه داده‌های بومی، پایگاه داده‌های رابطه‌ای، پایگاه داده NOSQL، پایگاه داده‌های مبتنی بر گراف و پایگاه داده‌های مبتنی بر جستجو و سیستم‌فایل‌ها هستند.

۱-۳-۲-۲- پایگاه داده‌های بومی[۱]

دراین نوع پایگاه داده‌ها ذخیره و شاخص‌گذاری عناصر RDF، از ابتدا و مخصوص داده‌های آنها انجام می‌گیرد. ذخیره RDF به صورت بومی برای پرس‌وجوهایSPARQL مناسب‌تر و موثرتر است و این به خاطر طراحی مناسب و مخصوص همین داده‌ها است. در طراحی سیستم بومیRDF3x[Neu10] ، عقیده بر این است که سیستم‌هایی که برای کاربردی خاص سفارش شده‌اند، می‌توانند نسبت به سیستم‌های عمومی بهتر عمل کنند. دلایلی که برای این امر می‌توان مطرح نمود:

۱- انتخاب ساختمان داده و الگوریتم مناسب همان داده به جای پشتیبانی از روشهای مختلف برای همه نوع داده همواره موثرتر است.

---

[۱]-Native

۲- سربار سیستم کم‌تر است.

۳- بهینه‌سازی سیستم داخلی ساده‌تر است و پیکربندی راحت‌تر صورت می‌گیرد و در نتیجه خود انطباقی سیستم نسبت به تغییرات محیط ممکن می‌شود.

۲-۳-۲-۲- پایگاه داده‌های رابطه ای[۱]

می‌توان بااستفاده از این نوع پایگاه‌داده ها از مزیت‌های چندین ساله آن‌ها استفاده نمود. البته برای کار با پایگاه‌داده‌های رابطه‌ای مانند SQL نیاز به لایه‌ی تبدیل SPARQL به SQL داریم. این نوع پایگاه داده‌ها برای RDFها، مناسب گزارش نشده‌اند که یکی از دلایل مهم آن می‌تواند عدم مقیاس‌پذیری در الحاق‌ها باشد[Med11].

۲-۳-۲-۳- پایگاه داده‌های NOSQL

NOSQL [NOSQL] یک عبارت کلی است و به معنای این است که از دسته‌ی پایگاه داده رابطه‌ای نیست که SQL را به عنوان زبان دسترسی اولیه خود پشتیبانی نماید. باید این نکته حتما ذکر گردد که پایگاه داده‌های NOSQL جایگزینی برای پایگاه داده‌های رابطه‌ای محسوب نمی‌شوند، بلکه به هدف توسعه این نوع پایگاه داده‌ها در حال رشد هستند. این نوع پایگاه داده معمولا توزیع شده هستند. مقیاس‌پذیری افقی دارند و شِمای ثابتی را تحمیل نمی‌کنند[Bug12].

سیستم‌هایNOSQL در قالب پایگاه‌داده‌های زیرقابل استفاده هستند:

- پایگاه داده کلید-مقدار
- پایگاه داده ستونهای خانواده (جدولی)
- پایگاه داده‌ای از گراف

---

۱ Relational DataBase

و....

HBase یک پایگاه داده توزیعی NOSQL و مبتنی بر ستون است که در بخش (۱-۲) به ویژگی-های این سیستم پرداخته شد. [Sun10]، [Fun12]، [Pap12]، [Kha12] برای ذخیره‌سازی شاخص‌های خود از این نوع پایگاه داده‌کمک گرفته‌اند.

۴-۳-۲-۲- پایگاه داده‌های مبتنی بر گراف

در این نوع پایگاه داده‌ها، داده‌ها در قالب گراف گونه خود ذخیره می‌شوند[Mac97] ، بدون اینکه شِمایی به داده‌ها تحمیل گردد. این نوع پایگاه داده‌ها هم برای داده‌های نیمه‌ساخت‌یافته و هم داده‌های با فرمت گرافی دیگر همچون داده‌های بیوانفورماتیک و .... وجود دارند[Agg10].

۵-۳-۲-۲- پایگاه داده‌های مبتنی بر جستجو

این نوع پایگاه داده برای پشتیبانی از مجاورت فیزیکی داده‌های مبتنی بر جستجو ظهور نمودند که در بخش ارزیابی این نوع مجاورت فیزیکی شرح داده می‌شود.

۶-۳-۲-۲- سیستم فایلها

سیستم‌فایلهایی همچون Hadoop می‌تواند به عنوان یک نامزد در ذخیره‌ی داده‌های حجیم RDF مورد استفاده قرارگیرد[Hadoop]. امکانات این سیستم فایل در بخش (۱-۲) تشریح شده‌است.

۴-۲-۲- معیارهای ارزیابی سیستم‌های ذخیره‌سازی و مدیریت شاخص‌ها

سیستم، بسته به نوع طراحی می‌تواند دارای امکاناتی باشدکه قوتی برای بازیابی و مقیاس‌پذیری هرچه بیشتر شاخص‌ها باشد و یا ضعفی در عملکرد آن ایجاد کند. ویژگی‌هایی که باید در این سیستم‌ها مورد ارزیابی قرار گیرد در ادامه شرح داده شده‌است. با توجه به اینکه پایگاه‌داده‌های بومی،NOSQL

ورابطه‌ای دراین زمینه بیشتر مورد مطالعه وبررسی قرار گرفته‌اند، این ویژگی‌ها برای این سه دسته پایگاه داده به طور مختصر بیان شده‌است و در مورد دو پایگاه دیگر این مقایسه صورت نمی‌گیرد.

ویژگی‌هایی که باید در انتخاب یک سیستم برای ذخیره‌سازی شاخص‌ها در نظر گرفته شود، عبارت‌اند از:

نحوه‌ی بهینه سازی پرس‌وجو توسط سیستم- در این زمینه، پایگاه‌های بومی‌مثل RDF3x [Neu10] و Jena TDB [Owe08]، بهترین عملکرد را دارند به این دلیل که فقط برای RDFها سفارشی شده‌اند. در زمینه بهینه سازی، پایگاه‌داده‌های بومی به ترتیب الحاق‌ها با استفاده از آمارهای به کارگرفته شده در بهینه‌سازی پرس‌وجو توجه خاصی می‌کنند که در کاهش هزینه الحاق بسیار موثر است[Har05].

شیوه الحاق روی داده‌ها- یک سری از پایگاه‌داده‌ها همچون پایگاه‌داده‌های رابطه‌ای در الحاق‌داده‌ها از عملیاتی استفاده می‌کنند که برای حجم گسترده‌ی RDFها مناسب نیستند[Wei08]. پایگاه‌داده‌های NOSQL و بومی در این زمینه می‌توانند بهتر عمل کنند. چرا که عمل ادغام در این پایگاه‌داده‌ها را می‌توان به صورت بهینه و مقیاس‌پذیرتر پیاده‌سازی نمود**[Sun10][Har07].**

شیوه اعمال توزیع شدگی در سیستم: پایگاه داده‌های بومی‌همچون Jena TDB و Yars2 به این امر توجه خاصی دارند. اما پایگاه‌داده‌های رابطه‌ای در مقیاس‌پذیری افقی به دلیل پیچیدگی بالا مناسب نخواهند بود. در مجموع باید گفت اگر حجم داده‌ها به این شکل در حال افزایش باشد مقیاس‌پذیری افقی شرط اصلی عملکرد یک سیستم مدیریت RDF خواهد بود که تا به امروز تنها پایگاه‌داده‌های NOSQL همچون HBase در این ویژگی موفق هستند[Med11].

شمای مورد پشتیبانی در مورد داده‌ها- پایگاه‌داده‌های NOSQL از هر نوع شِمایی به صورت غیرنرمال سازی شده، پشتیبانی می‌کنند که این امر در طراحی شِمایی مناسب برای RDFها یک قوت به

حساب می‌آید[Sun10]. در پایگاه‌داده‌های بومی‌نیز تمام اختیارات در دست طراح است[Har05]. در پایگاه داده‌های رابطه‌ای محدودیت در طراحی شِمای مختلف، یک نقص به‌حساب می‌آید[Med11].

مجاورت فیزیکی داده‌ها - در بین ویژگی‌های بیان شده، ویژگی مجاورت فیزیکی بیشتر از همه مورد بحث قرار گرفته‌است که در ادامه به شرح آن پرداخته می‌شود.

مجاورت فیزیکی یکی از مباحث مهم مطرح شده در پایگاه‌داده‌ها است [Wan10] [Aba07] [Wei08]. مجاورت داده به قرارگیری داده‌ها به صورت فیزیکی نسبت به هم، اشاره دارد. پایگاه داده‌ها بسته به نوع مجاورت فیزیکی داده‌ها به پایگاه داده‌ی مبتنی بر سطر، پایگاه داده مبتنی بر ستون و پایگاه داده مبتنی بر جستجو تقسیم می‌شوند. در ادامه این سه دسته مختصرا شرح داده می‌شود.

الف) پایگاه داده‌های مبتنی بر سطر[۱]: در پایگاه‌داده‌هایی که داده ها را مبتنی بر سطر ذخیره می‌کنند، داده‌های مربوط به یک سطر جدول، در فایل، پشت سر هم نوشته می‌شوند[Wil03][Bro03]. مشکل این نوع ذخیره‌سازی در این است که اگر در مدل پرس‌وجوی اعمالی روی داده‌ها، همیشه نیاز به بازیابی چند مشخصه باشد آنگاه تمام سطر بازیابی می‌شود که نیازی به آنها نیست. بنابراین ایده پایگاه-داده‌ی مبتنی بر ستون ارائه شد.

ب) پایگاه داده‌های مبتنی بر ستون:[۲] ایده‌ی اصلی در پشت پایگاه داده‌های مبتنی بر ستون، ذخیره جداول به عنوان مجموعه‌ای از ستونها است [Aba07][Wei08]. در این نوع ذخیره‌سازی که به ازای هر ستون، داده‌ها ذخیره می‌گردند، تنها ستونهایی بازیابی می‌شوند که لازم باشند. این نوع

---

[۱]-Row Oriented

[۲]Column Oriented

ذخیره‌سازی در شِمای مبتنی بر بخش بندی عمودی به کارگرفته شد و نتایج بسیار خوبی نیز به دست آمد. که دلیل این نتایج خوب در زیر شرح داده می‌شود.

مقایسه پایگاه داده‌های مبتنی بر سطر و پایگاه داده‌های مبتنی بر ستون[Aba07]:

ذخیره سرآیند تاپلها به صورت جداگانه: پایگاه‌داده‌ها به طور کلی ابرداده‌های هر تاپل را در ابتدای آن ذخیره می‌کنند. به عنوان مثال Postgres محتوی ۲۷ بایت سرآیند برای هر تاپل است که محتوی اطلاعاتی مثل زمان تراکنش درج، تعداد ویژگی‌ها در هر تاپل و فلگ‌های NULL است. در مقابل داده‌ها در جداول دو ستونه مبتنی بر ستون بیشتر از ۸ بایت نمی‌باشند، به خصوص زمانی که فشرده شده باشند. یک ذخیره‌ساز ستونی داده‌های مربوط به سرآیند را در ستونی جدا، نگه می‌دارد و می‌تواند حتی از آنها صرف نظر کند، زیرا اکثر آنها با این نوع ذخیره‌سازی قابل استفاده نیستند. بنابراین با توجه به اینکه اندازه‌ی هرتاپل برابر ۸ بایت و درPostgres برابر ۳۵ بایت است بنابراین پویش جدول در نوع ذخیره ستونی نسبت به Postgres چهار تا پنج برابر سریعتر می‌شود.

بهینه سازی برای تاپل‌های با اندازه‌ی ثابت[۲]: در ذخیره‌سازی مبتنی بر سطر اگر یک ویژگی با اندازه‌ی متغیر وجود داشته باشد اندازه‌ی کل تاپل متغیر می‌شود ولی در ذخیره سازی مبتنی بر ستون تنها ستونی با ویژگی متغیر، متغیر است. در ضمن در مبتنی بر ستون هر تاپل با اندازه‌ی ثابت در آرایه نگهداری می‌شود که می‌تواند برای ذخیره RDFها سود فراوان داشته باشد.

فشرده‌سازی یک ستون خاص: منظور از فشرده سازی، فشرده‌سازی داده‌های پایگاه‌داده‌های مبتنی بر ستون است. در ذخیره‌سازی مبتنی بر ستون، از آنجایی که هر ویژگی به صورت جداگانه ذخیره

---

[۱]-Tuple Header

[۲]-Fixed Length

می‌شود، هر صفت می‌تواند به شکل جداگانه‌ای فشرده شود و از الگوریتم‌های متناسب آن ستون استفاده شود.

ذخیره‌سازی مبتنی بر ستون برای زمانی مفید است که در پرس‌وجوها، کل یک سطر همیشه بازگردانده نشود و تغییر روی مقدار یک ستون برای کل رکوردها اعمال گردد. در مقابل ذخیره‌سازی مبتنی بر سطر برای زمانی مفید است که ستونهای یک رکورد با یکدیگر بازیابی می‌شوند و تغییر روی چند ستون قرار است همزمان اعمال گردد.

ج) پایگاه داده‌های مبتنی بر جستجو[۱] گاهی اوقات وضعیتی وجود دارد که نه پایگاه داده مبتنی بر سطر و نه پایگاه داده مبتنی بر ستون می‌توانند به خوبی عمل کنند. به عنوان مثال ممکن است کاربر در یک سایت در خواست یک وب سایت مبتنی بر RDF را داشته باشد که محتوی پایگاه‌داده‌ای از کتاب باشد. در طول درخواست، ممکن است کاربر از کتابها و مقالات به نویسندگان و کتابهای مرتبط با مقالات و کتابهای اولیه رجوع کند. بنابراین می‌توان ذخیره‌سازی را به نحوی انجام داد که رکوردهای R در کنار رکورهایی قرار داده شودکه از رکورد R معمولا به آن رکورد درخواست داده می‌شود. با این نحوه‌ی ذخیره‌سازی می‌توان هزینه I/O، و الحاق‌های زیاد را کاهش داد[Ver12].

مقیاس‌پذیری سیستم در برابر حجم سه‌گانه مورد پشتیبانی- در این زمینه، پایگاه‌داده‌های NOSQL و بومی‌توجه بهتری به این قضیه داشته‌اند و می‌توان از کد باز بودن و امکان توزیع شدگی این دو نوع پایگاه داده، به حد مطلوب، برای ذخیره شاخص‌های RDF به کار گرفت[Owe08][Sun10]. آزمایشات انجام گرفته در [Neu10] نشان داد که پایگاه داده‌های رابطه‌ای استفاده شده در Sesame و Jena در این زمینه دچار ضعف هستند. البته این ضعف به ویژگی مجاورت فیزیکی پایگاه‌داده‌ها نیز مربوط

---

[۱] Browsing Oriented Data Base

می‌شود. پایگاه‌داده‌های رابطه‌ای مبتنی بر سطر با شِمای یکپارچه مقیاس‌پذیری خیلی کمتری نسبت به پایگاه‌داده‌های مبتنی بر ستون با شِمای بخش‌بندی عمودی دارند[Aba07] [Wei08] .

به‌استناد تمام مطالب گفته شده در این فصل، می‌توان گفت پایگاه‌داده‌های مبتنی بر ستون(چه رابطه‌ای و چه NOSQL) و پایگاه‌داده‌های بومی امروزه از نظر ویژگی‌های مختلف به خصوص ویژگی‌های گفته شده در بالا مورد نقد و بررسی بسیاری از طراحان سیستم‌های بازیابی داده‌های RDF، قرار گرفته-است.

۵-۲-۲- خلاصه‌ی فصل

در این فصل ابتدا مفاهیم پایه‌ی پیش‌نیاز، جهت درک نحوه‌ی طراحی سیستم شاخص‌گذاری مبتنی بر پایگاه‌داده‌های NOSQL و یا فایل سیستم Hadoop مطرح شد. سپس در بخش کارهای مرتبط نمونه‌ای از سیستم‌های شاخص‌گذاری مختلف، هم در محیط متمرکز و هم در محیط توزیعی معرفی شدند. اکنون در این بخش تمام سیستم‌های شاخص‌گذاری RDF را از نظر شِمای شاخص‌گذاری و سیستم مورد استفاده جهت ذخیره شِمای مورد نظر، در جدول (۳-۲) و (۴-۲) خلاصه می‌کنیم.

جدول ۲-۳: مقایسه امکانات سیستم‌های مدیریت و بازیابی RDF

سیستم‌ها	سال	سیستم ذخیره‌سازی	شما	شاخص چند گانه	شاخص دیگر
[Wilo3] jena1	۲۰۰۳	NOSQL و رابطه ای	یکپارچه	S، P، O	-
Hive+HBase[JADID]	۲۰۱۳	NOSQL	یکپارچه	-	-
[Udr07] GRIN	۲۰۰۵	بومی	یکپارچه	-	✓
Yars2/swse [Har07][Hog11]	۲۰۰۷	بومی	یکپارچه	شش گانه	✓
[Wan10] CoDB	۲۰۱۰	رابطه ای	یکپارچه (مبتنی بر ستون)	شش گانه	-
jenaSDB http://jena. apache. org/	۲۰۰۶	رابطه ای	یکپارچه	شاخص ثانویه	-
[Wei08] HexsaStore	۲۰۰۸	رابطه ای	بخش بندی مبتنی بر همه عناصر(مبتنی بر ستون)	شش گانه	-
[Owe08] jenaTDB	۲۰۰۸	بومی	یکپارچه	شش گانه	-
Sesame /http://openrdf. org	۲۰۱۲	بومی	یکپارچه	پیش فرض: SPOCوPOSC	–

جدول ۲-۴: مقایسه امکانات سیستم‌های مدیریت و بازیابی RDF (ادامه)

سیستم‌ها	سال	سیستم ذخیره‌سازی	شما	شاخص چند گانه	شاخص دیگر
Sesame [Ses]	۲۰۱۲	اکثر ذخیره‌سازها	یکپارچه _ بخش بندی مبتنی بر Predicate	اختیاری	-
مبتنی بر فیلد [Mino8]	-	بومی	مبتنی بر نوع عنصر	-	-
Sixtuple_HBase [Sun10]	۲۰۱۰	NOSQL	بخش بندی مبتنی بر همه عناصر	شش گانه	-
RDF3x [Neu10][Neu11]	۲۰۰۹	بومی	یکپارچه	شش گانه	•
Sindice/siren [Del10]	۲۰۱۰	بومی	مبتنی بر گره	-	-
Jena_Habse [Kha12]	۲۰۱۲	NOSQL	ترکیبی	POS PSO SOP OPS	-
Structure index [Tra12]	۲۰۱۲	بومی	مبتنی بر ساختار	PEOS PESO POS PSO	-

فصل ۳- روش پیشنهادی

هدف از روش‌پیشنهادی، ارائه‌ی یک روش شاخص‌گذاری مبتنی بر موجودیت روی‌داده‌های RDF و با استفاده ار HBase است. قبل ار بررسی جزئیات سیستم، ساختار و امکانات کلی سیستم پیشنهادی در جدول (۱-۳) و (۲-۳) قابل مشاهده است. این امکانات در فصل قبل به طور مفصل شرح داده‌شد.

جدول ۳-۱: ویژگی کلی سیستم پیشنهادی

سیستم ا	سیستم ذخیره‌سازی	شِما	شاخص چند گانه	شاخص دیگر
سیستم پیشنهادی	NOSQL	مبتنی بر موجودیت	-	شاخص ساختار

جدول ۳-۲: ویژگی¬های شِمای پیشنهادی

شِما	نوع پرس‌و-جوی مورد حمایت	قدرت الحاق-ها	پشتیبانی از صفت‌های چندمقداری	شاخص چند گانه	روش دسترسی	هزینه بروزرسانی (درج)
مبتنی بر گراف	مبتنی بر موجودیت	حذف کامل الحاق	وجود دارد	–	Btree HBase	زمان تحلیل ساختار موجودیت+ زمان شاخص‌گذاری

۱-۳- انگیزه

۱-۱-۳- انگیزه ارائه الگوریتم جدید مبتنی بر موجودیت

روش‌های پیشین که مبتنی بر موجودیت هستند، یا مستندسازی دقیقی ندارند [wil06] و یا الگوریتم آنها در برابر حجم بالای داده، خروجی مطلوب را تولید نمی‌کند و وابسته به جداول رابطه‌ای هستند[Lev09]. آخرین روش‌پیشین که مبتنی بر موجودیت و مشابه کار فعلی است، روش [Lev09] است. در این روش هدف، خوشه‌بندی موجودیت‌ها و قرار دادن آنها در جداولی مشترک است. با توجه به حضور مقادیر *NULL* و هزینه‌ی الحاق در جداول رابطه‌ای، هدف از الگوریتم روش [Lev09] کاهش مقادیر *NULL* و هزینه الحاق است. اما هرچه هزینه‌ی الحاق پایین آورده شود و ستون‌ها بیشتر شود

مقادیر *NULL* افزایش می‌یابد؛ از این رو هدف اصلی روش [Lev09]، ایجاد تقابل بین تعداد مقادیر *NULL* و هزینه الحاق است. اما ارزیابی‌های اولیه در بخش(۳-۲-۴) و روی سیستم مورد نظر نشان داد که این سیستم، در برابر مجموعه‌داده‌ای با حجم بالا به دلیل کم شدن میزان حمایت الگوها و نیزعدم در نظر گرفتن همپوشانی، به خوبی عمل نمی‌کند. بنابراین هدف از روش پشنهادی ارائه یک روش موثر برای خوشه‌بندی داده‌ها و البته در حجمی بالاتر است.

۲-۱-۳- انگیره‌استفاده از HBase

اگرچه پایگاه‌داده‌های رابطه‌ای با ترکیبی از ویژگی‌ها همچون سادگی، انعطاف‌پذیری، عملکرد بالا، مقیاس‌پذیری و سازگاری خود را به کاربران اثبات کرده‌اند، اما ضرورتا از روشهای جایگزین که تنها متمرکز به یکی از این ویژگی‌ها هستند، بهتر عمل نمی‌کنند.

با توجه به اینکه ویژگی اصلیِ ابر محاسباتی در دسترس‌پذیر بودن آن است، سرمایه گذاران به فکر رفع این محدودیت شدند. بنابراین برای ایجاد یک فضای مقیاس‌پذیر جهت ذخیره داده‌ها، هدف طراحی سیستمی‌شد که تمرکز ویژه روی مقیاس‌پذیری داشته باشد.

پایگاه‌داده‌های رابطه‌ای برای پردازش در سطح تراکنش مناسب هستند؛ اما برای پردازش در سطح داده‌های زیاد مناسب نخواهند بود. این امر به این دلیل است که نیاز به پویش[۱]حجم گسترده-ای از جداول دارد. همچنین حالت انتظار و بن بست[۲]در این سیستم‌ها با افزایش تراکنش و همزمانی-ها افزایش می‌یابد. از این رو، عامل اصلی این مشکلات یعنی نرمال‌سازی حذف می‌شود. با حذف نرمال‌سازی مشکل الحاق و مشکلات تراکنش نیز که حالت انتظار و بن بست دارد حل می‌شود. با این رویکرد، پایگاه‌داده‌های NOSQL ایجاد شدند.

۱- Scan

۲ Wait and DeadLock

مزیت اول این نوع از پایگاه داده‌ها، مناسب بودن آنها برای محیط ابری است. چون این نوع پایگاه‌ها به راحتی و به صورت پویا مقیاس‌پذیرند، می‌توانند برای سرمایه گذارانی که وب‌سرویسهای چند کاربره[۱]را ارائه می‌دهند، بهترین گزینه باشد. این نوع پایگاه‌داده‌ها یک ساختار ذخیره داده‌ی نسبتا ارزان و با پتانسیل بالای مقیاس‌پذیری را مهیا می‌کند.کاربران در قبال آنچه که استفاده می‌کنند پول پرداخت می‌کنند اما با توجه به افزایش نیازهایشان، استفاده آنها نیز بیشتر می‌شود. بنابراین سرمایه گذاران می‌توانند بدون هیچ محدودیتی در مقیاس‌پذیری، با توجه به حجم کاربران، مقیاس-پذیری کل سیستم را بالا ببرند[RDB].

اما مشکلات این نوع پایگاه‌ها در جای خود باقی است. پایگاه‌داده‌های رابطه‌ای در پایین‌ترین سطح، به کاربر، اطمینان جامع بودن[۲]داده‌ها را می‌بخشند اما در پایگاه داده‌های NOSQL این امکان وجود ندارد و باید توسط کاربران در برنامه‌های کاربردی نوشته شود که نوشتن این برنامه‌ها بدون خطا نیستند؛ بنابراین نمی‌توان گفت که NOSQLها به عنوان جایگزینی برای جداول رابطه‌ای هستند؛ بلکه بسته به عواملی از جمله مدل داده‌ی سیستم مورد نظر، مدل سازگاری، مدل فیزیکی(توزیع شدگی سیستم) قدرت خواندن و نوشتن، تحمل پذیری سیستم در خطا، توازن بار، قفلها و حالت‌های انتظار، باید سیستم مورد نظر انتخاب شود[Med11].

پایگاه داده‌های رابطه‌ای مقیاس‌پذیری خوبی دارند اما تا زمانی که فقط در یک گره این مقیاس‌پذیری انجام گیرد. زمانی که حجم داده در یک گره بالا می‌رود و به حد نهایی خود می‌رسد نیاز به توزیع داده‌ها در بین چند گره‌است. این مرحله زمانی است که پیچیدگی‌های پایگاه‌های رابطه-ای بسیار بالا می‌شود. حال اگر مقیاس‌پذیری تا هزاران هزار گره نیاز باشد، قابلیت آنها بسیار پایین

---

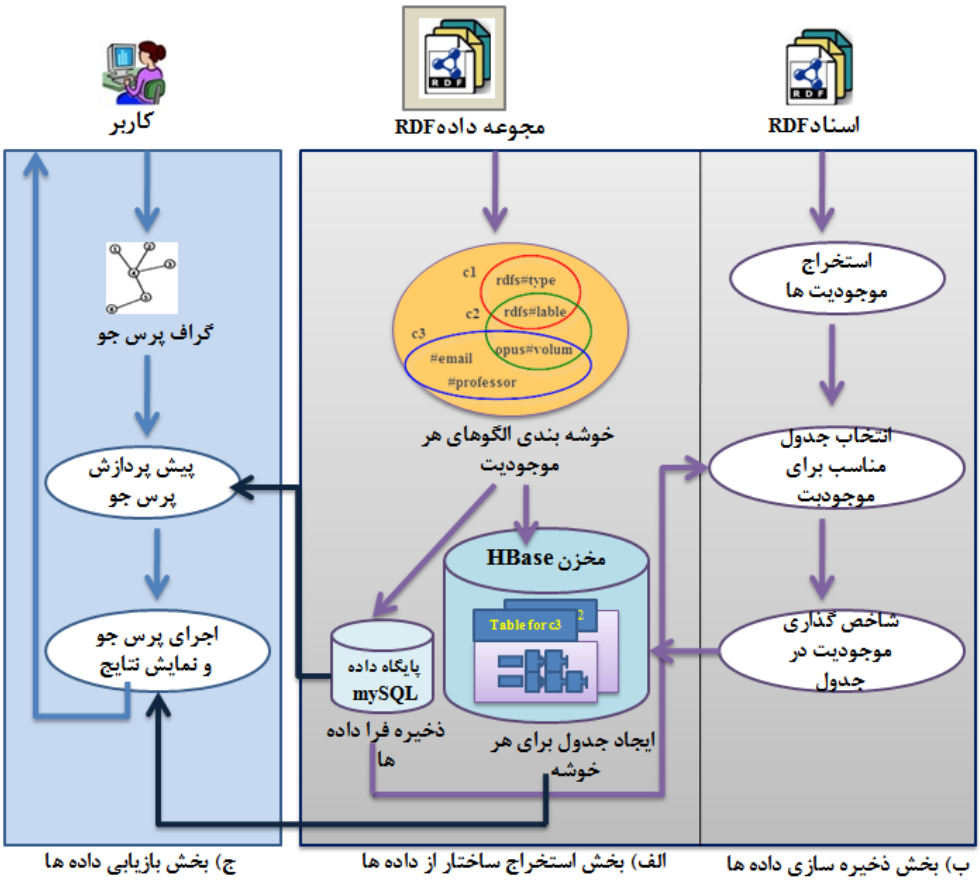
[۱]-MultiUser

[۲]- Integrity

می‌آید به نحوی که به علت وجود تراکنش‌ها، دسترس پذیری آنها در یک شبکه توزیعی تحت تاثیر قرار خواهد داد[RDB].

۲-۳- معماری پیشنهادی

معماری پیشنهادی این مقاله که در شکل(۱-۳) نشان داده شده‌است، از سه بخش اصلی، شامل: استخراج ساختار، ذخیره‌سازی و بازیابی داده‌ها تشکیل می‌شود. در ادامه این سه بخش و اجزای درون آن شرح داده می‌شود.



شکل ۳-۱: معماری سیستم پیشنهادی

۱-۲-۳- استخراج ساختار از داده‌ها

این بخش در پی یافتن ساختار، در بین داده‌های نیمه‌ساخت یافته‌است. ساختار استخراجی در نهایت، شِمای ذخیره‌سازی داده‌ها را تشکیل می‌دهد. این واحد در دو فاز خوشه‌بندی الگوهای هر

موجودیت و ایجاد شِما در جداول HBase، عملیات لازم را انجام می‌دهد. هر یک از این دو فاز در ادامه شرح داده می‌شود.

۱-۱-۲-۳- خوشه‌بندی الگوهای هر موجودیت

در این فاز ابتدا الگوهای موجود برای هرکدام از موجودیت‌ها استخراج می‌شوند و سپس با هدف شناسایی الگوهای مشابه، الگوریتم خوشه‌بندی پیشنهادی اجرا می‌شود. در ادامه روند اجرای این الگوریتم شرح داده می‌شود.

به مجموعه‌ای از صفات که یک موجودیت را توصیف می‌کنند الگو گفته می‌شود. به عنوان مثال یک یا چند موجودیت ممکن است از الگوی A که در زیر نوشته شده و دارای سه عضو است، جهت توصیف ویژگی‌های خود استفاده نمایند.

A={creator,title,abstract}

در این الگوریتم هدف، یافتن موجودیت‌هایی است که بیشترین شباهت را با یکدیگر از نظر الگو دارند. روند اجرای این الگوریتم شامل ۳ گام است که در ادامه هر گام به تفکیک، شرح داده می‌شود.

گام اول- تشخیص الگو) در این فاز کل مجموعه‌داده‌ی RDF مورد پردازش قرار گرفته و برای هر موجودیت الگوی صفاتش به همراه تعداد تکرار آنها کشف می‌شود. سپس از بین این الگوها، الگوهای تکراری حذف و مابقی الگوها به خروجی فرستاده می‌شود.

گام دوم- اعمال الگوریتم) بعد از تشخیص الگوها، نوبت به اعمال یک الگوریتم خوشه‌بندی مناسب، جهت خوشه‌بندی الگوها است. برای این منظور روش خوشه‌بندی متراکم[۱][Clus] متناسب با کارفعلی، مورد استفاده قرار می‌گیرد. این الگوریتم خوشه‌ها را به شیوه‌ی پایین به بالا و بر اساس معیار شباهت پیشنهادی می‌سازد.

---

۱ Agglomerative

روند انجام این الگوریتم روی الگوهای استخراجی به این ترتیب است:

۱) در ابتدا ماتریس فاصله را (از نظر معیار شباهت که برای تشابه الگوها طراحی شده‌است) به شکل یک آرایه‌ی دو بعدی می‌سازد. سپس هر الگو را در یک خوشه مجزای خودش قرار می‌دهد.

۲) در بین خوشه‌ها، خوشه‌هایی را که بیشترین شباهت و یا در واقع کمترین فاصله رادارند، انتخاب می‌کند.

۳) این دو خوشه را با یک خوشه جدید متشکل از عناصر این دو خوشه جایگزین می‌کند.

۴) فاصله و یا در واقع شباهت دیگر خوشه‌ها را با این خوشه جدید، مجددا حساب و ماتریس فاصله را بروز می‌کند.

۵) مجددا به مرحله ۲) می‌رود و این کار را تازمانی که شباهت دو خوشه انتخابی برای ادغام از حد آستانه β بالاتر نرود، تکرار می‌کند. حد آستانه β تعیین کننده‌ی تعداد خوشه خروجی است.

برای اندازه‌گیری شباهت[۱]موجودیت‌ها، فرمول (۱-۳) ارائه شده‌است. اگر فرض گردد، هر صفت با p1 و p2 و... نام گذاری شود آنگاه می‌توان دو الگوی صفت A و B را به فرم ریاضی زیر نشان داد.

$$A=\{p_i \mid i \in N , p_i \in \text{Predicates}\}$$

$$B=\{p_j \mid j \in N , p_j \in \text{Predicates}\}$$

بنابراین شباهت این دو الگو از رابطه‌ی زیر به دست می‌آید.

[۱] Similarity

$$Similarity = \alpha.\left|\frac{common(A,B)}{L(A)} - \frac{common(A,B)}{L(B)}\right| + \frac{1}{\frac{common(A,B)^2}{L(A).L(B)}}$$

فرمول(۱-۳)

common (A,B): تعداد صفات مشترک موجود در A و B است.

L: تعداد صفاتی است که عضو هر مجموعه مورد نظر است.

α: ضریب همبستگی بین الگوها است.

در این فرمول با عمل تقسیم صفات مشترکِ هر الگو بر تعداد اعضای آن، شباهت هر دو الگو نسبت به هم سنجیده می‌شود. بنابراین هرچه تفاوت موجود در قدر مطلق کم‌تر باشد، می‌توان گفت دو الگو از نظر تعداد اعضا مشابه یکدیگر هستند و البته از نظر تعداد اعضای مشترک ادعایی نمی‌توان داشت. بنابراین ضریب آلفا برای محاسبه میزان اثر این تفاوت در نظر گرفته شده‌است. به جهت اینکه بتوان الگوهای بزرگتر، هم از نظر تعداد کل اعضا و هم تعداد اعضای مشابه را در یک خوشه قرار داد، نسبت تشابه به اندازه هر یک در یکدیگر ضرب می‌شود. حال هرچه حاصل ضرب بزرگتر و حاصل تفریق کوچکتر باشد احتمال ادغام دو خوشه‌ی کاندید بیشتر از بقیه خوشه‌ها است.

بررسی یک مثال) به عنوان مثال فرض شود برای سه الگوی A، B و C قرار است شباهتشان نسبت به هم سنجیده شود.

A={ #paper_title,#publisher, #isbn }

B={#author, #paper_title, #year, #volume, #publisher, #isbn}

C={ #author, #paper_title, #abstract,}

اگر مقدار α برابر ۵ در نظر گرفته شود در نتیجه شباهت ببین A,B به طریق زیر می‌شود. بین A,B همانطور که مشهود است ۳ صفت مشترک است.

common (A,B)=3

$L(A)=3, L(B)=6$

$\alpha=5$

با جایگذاری در فرمول (۱-۳) شباهت محاسبه می‌شود.

$$Similarity(A,B)=5*|(3/3)-(3/6)|+1/(3^2/6*3)=4.5$$

شباهت ببین A,C نیزبه این طریق می‌شود که :

$common\ (A,C)=1$

$L(A)=3, L(C)=3$

$\alpha=5$

با جایگذاری در فرمول (۱-۳) شباهت محاسبه می‌شود.

$$Similarity(A,C)=5*|(1/3)-(1/3)|+1/(1/3*3)=9$$

با توجه به اینکه A وB کمترین فاصله را دارند، بنابراین بیشترین شباهت را نسبت به A وC دارند.

گام سوم- ایجاد نماینده خوشه‌ها) بعد از ایجاد خوشه‌ها نوبت به انتخاب نماینده‌ی‌خوشه‌ها است. نماینده‌ی خوشه‌ها مهمترین نقش را در فرایند ذخیره و بازیابی موجودیت‌ها دارند. در واقع نمایندگان خوشه‌ها تداعی کننده‌ی ساختار یک مجموعه‌داده هستند. بنابراین عملیات تشکیل این سرخوشه‌ها بسیار مهم و حساس است.

بدیهی است که نماینده‌ی خوشه، متشکل از تمام صفات موجود در الگوهای هر خوشه است. اما در این حین ممکن است برای یک خوشه نماینده‌ای ظاهر شود که الگویی مانند x از یک خوشه دیگر، به این نماینده نسبت به نماینده‌ی خود، شباهت بیشتری داشته باشند. منظور از شباهت بیشتر، این است که نماینده مورد نظر با تعداد اعضای (صفات) کمتری، می‌تواند الگویی مانند x را پشتیبانی کند. بنابراین عملیات یافتن بهترین سرخوشه برای اعضای خوشه‌ها صورت می‌گیرد. این عملیات به این ترتیب است که:

۱) برای هر خوشه تشابه صفات آن را با نماینده آن خوشه و نمایندگان دیگر خوشه‌ها حساب می‌کند.
۲) اگر نماینده‌ای مناسب‌تر برای آن عضو پیدا شد،
- ابتدا آن الگو را از درون خوشه‌ی قبلی حذف می‌کند و مجددا برای آن خوشه نماینده جدید می‌سازد.
- آن الگو را به خوشه جدید منتقل می‌کند.

۳) این کار تازمانی صورت می‌گیرد که هیچ الگویی از درون خوشه‌ها تمایل به عضویت در خوشه‌ای با نماینده‌ی مشابه‌تر نداشته باشند.

۲-۱-۲-۳- ایجاد شِما

در این بخش برای هر نماینده‌ی خوشه‌ی ایجاد شده، یک جدول از HBase ایجاد می‌شود. با توجه به خاصیت غیرنرمال بودن جداول HBase، طراحی یک شِمای کارامد نقش اصلی را در عملکرد HBase ایفا می‌کند. HBase در قبال این خاصیت امکان ایجاد ستون‌های زیاد، نسخه‌بندی و ایجاد کلید هوشمند را ارئه داده است. برای این منظور ابتدا توصیفی از ساختار جداول بیان می‌شود.

هدف از راهکار پیشنهادی، ذخیره موجودیت‌های مشابه در یک جدول است که هر خوشه از الگوریتم خوشه‌بندی، نشان دهنده‌ی یک الگو از صفات است. نماینده‌ی هر خوشه نیز به نحوی بیانگر صفاتی است که الگوهای عضو این خوشه مورد حمایت قرار می‌دهند. از این رو هر نماینده‌ی خوشه، تشکیل یک جدول از HBase را می‌دهد. ستون‌های این جدول صفات درون نماینده‌ی خوشه‌ها است.

بنابراین فراداده، نگاشتی از نماینده‌ی خوشه‌ها به جدولی در HBase است. علت ذخیره این داده‌ها در MySQL این است که حجم فراداده‌ها بسیار کم است و قرار دادن آن در HBase کارامد نخواهد بود.

با توجه به توضیحات داده‌شده در بخش(۱-۲) مدل انتخابی برای طراحی جداول HBase، مدل ستون‌های زیاد با سطرهای کم است. مشکل اصلی این نوع الگوریتم حضور داده‌های پراکنده است که هزینه‌ی جستجو برای رسیدن به داده‌های مورد نظر را افزایش می‌دهد. در روش پیشنهادی با توجه به الگوریتم خوشه‌بندی این مشکل حل شده و موجودیت‌های مشابه در یک جدول ذخیره می‌شوند. در این روش‌پیشنهادی توانسته‌ایم امکان دسترسی دقیق‌تر را به جداول برای هر موجودیت ایجاد نماییم.

اما به جهت اینکه مزیت مدل طراحی دیگر HBase (ستون‌های کم با سطرهای زیاد)که امکان جست‌وجوی پیشوندی است از دست نرود تدابیری در شِمای ارائه شده، اندیشیده‌ایم.

در هنگام ایجاد خوشه‌ها به هر نماینده‌ی خوشه و اعضای آن به طور جداگانه یک شناسه تعلق می‌گیرد. هنگام شاخص‌گذاری، برای هرموجودیت که الگوی مورد نظر آن کشف شد، شناسه‌ی خوشه و نماینده‌ی خوشه‌ای که متعلق به آن است بازیابی می‌شود. سپس موجودیت مورد نظر به همراه ترکیبی از شناسه‌ی این دو در جدول ذخیره می‌شود. به عنوان مثال برای یک موجودیت مقاله‌ی A که دارای الگوی p است کلید سطری مانند K مطابق زیر ساخته می‌شود.

P={#author, #abstract, #title}

K=<شناسه‌ی نماینده‌ی (p)+شناسه‌ی (p)> + A

بنابرانین در حالت کلی، جدول HBase ایجاد شده برای یک خوشه مانند c1 که دارای شش صفت است، دارای ساختاری همچون شکل(۲-۳) است.

htable1						
RowKey	“CF”					
< HeadClusterId + SubClusterId > +Entity URI	p1	p2	p3	p4	p9	p16
…						

شکل ۳-۲: یک نمونه از جدول HBase برای خوشه‌ای مانند c1 با شش صفت

با این نوع طراحی به مزیت‌هایی از جمله:

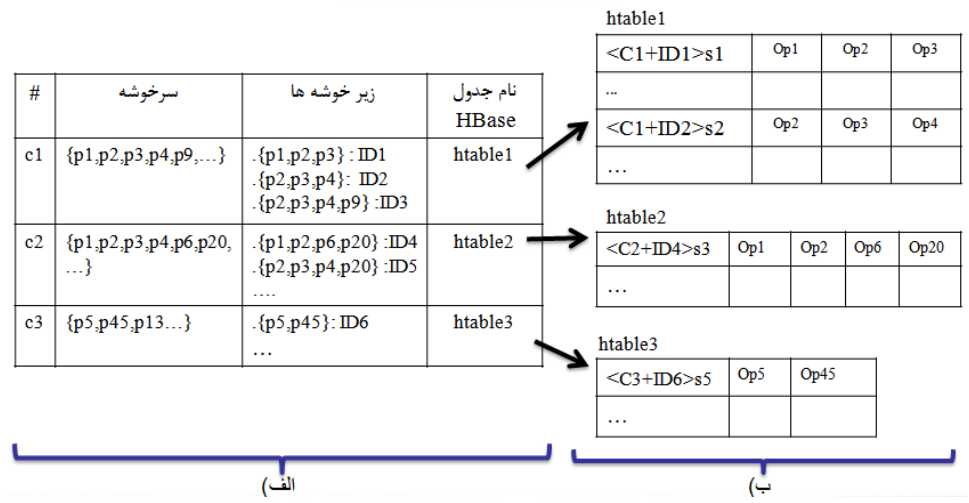
۱) ذخیره‌سازی موجودیت‌های یک خوشه در کنار هم که منجر به بازیابی موثر داده‌های مشابه از نظر ساختار می‌شود. چرا که HBase در خواندن داده‌ها به صورت ترتیبی نسبت به تصادفی سریعتر عمل می‌کند[Fun12].

۲) امکان جست‌وجوی پشوندی روی سطرها با سرعت بالا

۳) کوچکتر کردن فضای جست‌وجو برای بازیابی دقیق یک موجودیت با تعریف جداول جداگانه متناسب هر الگو

با این توضیحات، تفاوت شِمای ارائه شده با شِمای تک جدولی [JADID] نیز مشهود است.

در شکل (۳-۳) می‌توان به طور دقیق‌تر نحوه‌ی ذخیره موجودیت‌ها را نشان داد. هر موجودیت به همراه پیشوند متناسب خود، یک کلید سطر از جدول را تشکیل داده‌است. مقادیر مربوط به هر صفت یعنی Objectها نیز در درون ستون‌های مربوطه ذخیره می‌شوند.



شکل ۳-۳: الف) فرا داده‌ها در MySQL . ب) جداول HBase

در ضمن با توجه به اینکه در هنگام ذخیره‌سازی در قالب مدل فیزیکی HBase، نام ستون و ستون‌خانواده دائما به ازای هر کلید-مقدار ذخیره می‌شود بنابراین نام‌های هر ستون و ستون‌خانواده کوچک در نظر گرفته تا مقایسه سریعتر باشد.

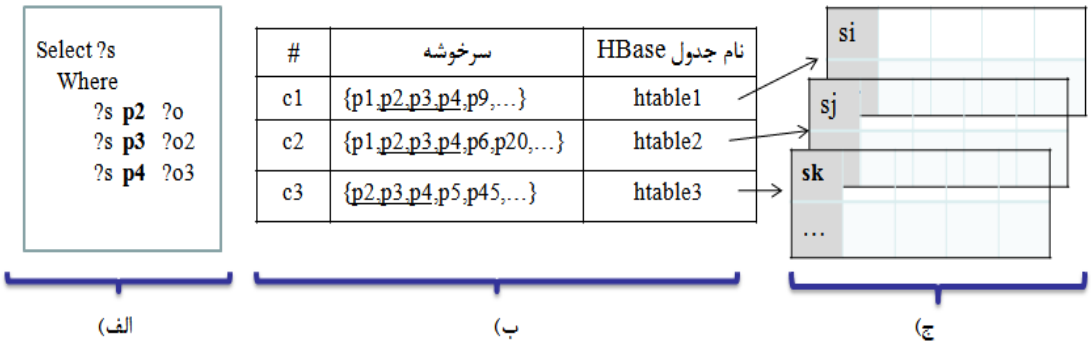
۳-۲-۲- ذخیره‌ی داده‌ها

در این بخش، اسناد به تفکیک مورد پردازش قرار می‌گیرند تا یک موجودیت با الگوی صفات آن استخراج شود. سپس با توجه به فراداده‌های ذخیره شده در واحدِ استخراجِ ساختار، مشابه‌ترین نماینده‌ی خوشه انتخاب و نامِ جدولِ HBase مورد نظر جهت ذخیره‌سازی موجودیت بازیابی می‌شود.

۳-۲-۳- بازیابی‌داده‌ها

با توجه به اینکه خواندن ترتیبی در HBase با سرعت بالاتری انجام می‌گیرد؛ داده‌ها با استفاده از صافی HBase به سرعت، مورد دسترسی قرار گرفته و به صورت ترتیبی از حافظه خوانده می‌شوند.

کاربر در پرس‌وجو به دنبال یک موجودیت با الگویی از صفات است. از این رو در این بخش ابتدا پرس‌وجو، مورد پردازش قرار می‌گیرد. و با استخراج این الگو از پرس‌وجو، عملیات یافتن نمایندگانی از خوشه‌ها که این الگو را حمایت می‌کنند آغاز شده و به ازای هرنماینده‌ی خوشه، به جدول مورد نظر، جهت بازیابی داده‌ها دسترسی صورت می‌گیرد. با توجه به شکل(۴-۳) ابتدا الگو از پرس‌وجوها استخراج می‌شود(شکل۳-۴- الف)، سپس الگوی مورد نظر در ابر داده‌ها جست‌وجو می‌شود تا نام جدول HBase مورد نظر استخراج (شکل۳-۴- ب) و در نهایت داده‌ها از جدول بازیابی شوند.



شکل ۳-۴: گام‌های پردازش پرس‌وجو در سیستم پیشنهادی: الف) پردازش پرس‌وجو ب) استخراج خوشه‌های مطابق با پرس‌وجو از MySQL به همراه نام جداول HBase ج) دسترسی به جداول HBase جهت بازیابی

همانطور که مشخص است خوشه‌های ایجاد شده به نوعی نشانگر ساختار قرارگیری داده‌ها در یک مجموعه‌داده‌ی RDF هستند. با در نظر گرفتن این مساله می‌توان هم مزایای گفته شده اعم از حذف سربار *NULL* در ذخیره‌سازی داده‌ها و حذف الحاق در بازیابی را داشت و هم کاربر را نیز از این ساختار واقعی داده‌ها آگاه نمود. تحقیقات مختلفی در کارهای پیشین وجود دارد که به کاربر در اعمال پرس‌وجوهای اسپارکل روی داده‌ها کمک می‌کنند[LOG]. این روش‌ها با پردازش نگاره‌ی ضبط شده در پرس‌وجوهای قبلی کاربران، الگوهایی از پرس‌وجوها را که بیشتر رخ داده، استخراج می‌کنند. سپس پرطرفدارترین صفات را متناسب با پرس‌وجوی کاربر به او پیشنهاد می‌دهند. در راهکار پیشنهادی فرایند کمک به کاربر بر اساس داده‌های واقعی ذخیره شده در HBase صورت می‌گیرد. مزایای این روش عبارت‌اند از:

- کمک به کاربران و مدیرانی که قصد آگاهی یافتن از ساختار و مدل واقعی داده‌های یک مجموعه‌داده‌ی RDF را دارند.
- قابل استفاده در سیستم‌هایی که، پرس‌وجوهای کاربران در آن موردی بوده و بنابراین پردازش نگاره و استخراج الگو هزینه‌بر است.

در این روش، کاربر ابتدا الگوی اولیه‌ی مورد نظر خود را وارد نموده و سپس تقاضای الگویی را می‌کند که اولا اکثر موجودیت‌ها از آن الگو جهت توصیف خود استفاده کرده‌اند و ثانیا الگوی مورد

نظرش را نیز پشتیبانی می‌کند. سیستم نیز با توجه به فراداده‌های ذخیره شده، در خوشه‌ها به دنبال الگویی با این شرایط می‌گردد. بعد از بازیابیِ الگوی مورد نظر، جهت ارائه دید بهتر از وضعیت صفت-های درون الگوی بازگشتی، صفات دسته‌بندی می‌شوند. دسته‌ی اول، صفات رایجی هستند که در مجموعه‌داده‌ی مورد نظر، معمولا در همه‌ی الگوها حضور دارند و در واقع عام هستند، و دسته‌ی دوم، صفاتی هستند که کمتر در الگوهای دیگر حضور دارند و خاص هستند.

به عنوان مثال فرض شود که کاربر می‌خواهد با نحوه‌ی رایج تعریف یک موجودیت مقاله، در یک مجموعه‌داده آشنا شود تا بتواند یا در این مجموعه‌داده، موجودیت جدید تعریف کند و یا بتواند پرس‌وجوهای دقیق‌تر را با توجه به ساختار داده‌ها به سیستم اعمال کند. بنابراین الگوی P={#author,#paper_title} را به سیستم اعمال می‌کند و از سیستم تقاضای پیشنهاد صفات دیگری را می‌کند که با این دو صفت رخ داده‌اند. بنابراین سیستم در ابر داده‌ها به دنبال پرطرفدارترین الگویی می‌گردد که الگوی p درون آن باشد. الگوی پیشنهادی سیستم، ممکن است الگویی مانند الگوی B باشد که نشان می‌دهد بیشترین موجودیت‌های کتاب، با این الگو توصیف شده‌است.

B={#author, #paper_title, #year, #volume, #publisher, #isbn}

همچنین برای ارائه یک دیدگاه بهتر از وضعیت صفات، دو صفت خاص پیشنهادی از درون الگوی B به ترتیب isbn# و publisher# و دو صفت عام پیشنهادی، type# و year# است. بنابراین این دیدگاه به فرد منتقل می‌شود که در جست‌وجوی دقیق موجودیتِ مقاله، اولا از چه صفاتی می-تواند کمک بگیرد و ثانیا جهت تعریف یک موجودیتِ مقاله، رایج‌ترین نوع تعریف را در دسترس دارد.

۳-۳- خلاصه‌ی فصل

در این فصل ابتدا به طور دقیق‌تر انگیزه‌ی اصلی از ارائه‌ی الگوریتم خوشه‌بندی مبتنی بر موجودیت روی جداول HBase مطرح شد تا تفاوت روش پیشنهادی با دیگر روش‌های پیشین پررنگ شود. سپس با معرفی معماری روش پیشنهادی، اجزای سیستم به طور کامل شرح داده‌شد.

فصل ۴- پیاده‌سازی و ارزیابی

سیستم پیشنهادی به زبان برنامه‌نویسی Java پیاده‌سازی شده‌است. در این فصل ابتدا در بخش (۴-۱) روند پیاده‌سازی سیستم پیشنهادی شرح و در بخش(۴-۲) مورد ارزیابی قرار می‌گیرد.

۴-۱- پیاده‌سازی سیستم پیشنهادی

همانطور که از معماری سیستم پیشنهادی در بخش (۲-۳) پیداست، سیستم مورد نظر دارای سه بخش استخراج ساختار از داده‌ها، بخش شاخص‌گذاری داده‌ها و بخش بازیابی است. روند پیاده-سازی و ارزیابی نیز بر مبنای این سه‌بخش شرح داده می‌شود.

۴-۱-۱- پیاده‌سازی بخش استخراج ساختار از داده‌ها

این بخش دارای دو فاز خوشه‌بندی موجودیت‌ها و ایجاد شِمای جداول در HBase است. پیاده‌سازی هریک از این دو فاز در ادامه بیان می‌شود.

۴-۱-۱-۱- خوشه‌بندی موجودیت‌ها

الگوریتم خوشه‌بندی ارائه شده دارای سه گام است که مراحل پیاده‌سازی هر یک در ادامه شرح داده‌می‌شود.

گام اول) استخراج الگو

برای تشخیص الگو دو راهکار وجود دارد. راهکار اول استفاده از پارسرهایی همچون Jena است که با توجه به اینکه حجم داده‌ها زیاد است انجام این کار در درون حافظه‌ی اصلی و به صورت متمرکز امکان‌پذیر نیست. در این بخش، جهت کاهش هزینه‌ی تشخیص الگو، از مدل برنامه‌نویسی توزیعی

Hadoop یعنی نگاشت-کاهش با دو واحد کاری[۱] استفاده شده‌است. واحد کاری اول، الگوهای هر موجودیت را استخراج می‌کند و سپس واحد کاری دوم الگوهای منحصر به فرد را به علاوه تعداد رخداد آنها در خروجی نمایش می‌دهد.

واحد کاری اول- شبه‌کد این واحد کاری در شکل(۱-۴) قابل مشاهده‌است. این واحد کاری در پی یافتن الگوهای هر موجودیت است. برای این منظور سه‌گانه‌ها در واحد نگاشت پردازش می‌شوند. سپس با استفاده از پارسر Sesame[sesame]، هر موجودیت به عنوان کلید و صفت آن به عنوان مقدار به خروجی داده می‌شود. در مرحله کاهش، با توجه به خروجی بخش نگاشت، ورودی هریک از موجودیت‌ها، به عنوان کلید و صفات آن به عنوان مقدار است. در این مرحله هر موجودیت به همراه لیستی از صفات آن در اختیار است. با توجه به اینکه هدف فقط استخراج الگوهاست، بنابرین الگوهای هر موجودیت به عنوان کلید و عدد "۱" به عنوان مقدار جهت محاسبه‌ی تعداد این الگوها (در واحد کاری دوم)، در خروجی چاپ می‌شود.

```
1. map(key,value)
2. //key:irrelevant
3. //value: triples in N-triple Format
4. Subject=getSubject(value)
5. Predicate=getPredicate(value)
6. Context.Write(Subject,Predicate)

1. Reduce(Key,iterator Values)
2. String Pattern=""
3. for(value in Values)
4.         Pattern.append(value+ " Splitable Character ")
5.         //"Splitable Character" is Charechter to seprate predicates in Pattern
6. Context.Write(Pattern,int(1))
```

شکل ۴-۱: شبه‌کد واحد کاری اول جهت استخراج الگو

واحد کاری دوم- شبه‌کد این واحد کاری در شکل(۲-۴) قابل مشاهده‌است. در این واحد کاری،

[۱] Job

هدف یافتن الگوهای منحصر به فرد، بعلاوه یک آمار کلی از تعداد آنها در مجموعه‌داده مورد نظر است. بنابراین خروجی واحد کاری اول به عنوان ورودی این واحد کاری قرار می‌گیرد. در مرحله نگاشت، ورودی، الگوهای استخراجی به عنوان کلید و عدد "۱" به عنوان مقدار است. در نگاشت، پردازشی صورت نمی‌گیرد و مستقیما داده‌ها به خروجی منتقل می‌شوند. در مرحله کاهش، به ازای الگوهای تکراری ، تعداد تکرار آن (تعداد اعداد "۱") شمارش و در خروجی چاپ می‌شود. این تعداد تکرار به نحوی تداعی کننده‌ی تعداد موجودیت‌هایی است که با آن الگو ظاهر شده‌اند.

```
1: map(key,value)
2: //key: is out put key  of Job1: Pattern
3: //value: : is out put value of Job1: 1
4: //doing nothing and only print to output
5: Context.Write(key,value)

1: Reduce(Key,iterator Values)
2: //only one key go to the output
3: int sum=0
4: for(value in Values)
   addToSum(sum,value)
5: Context.Write(Key,sum)
```

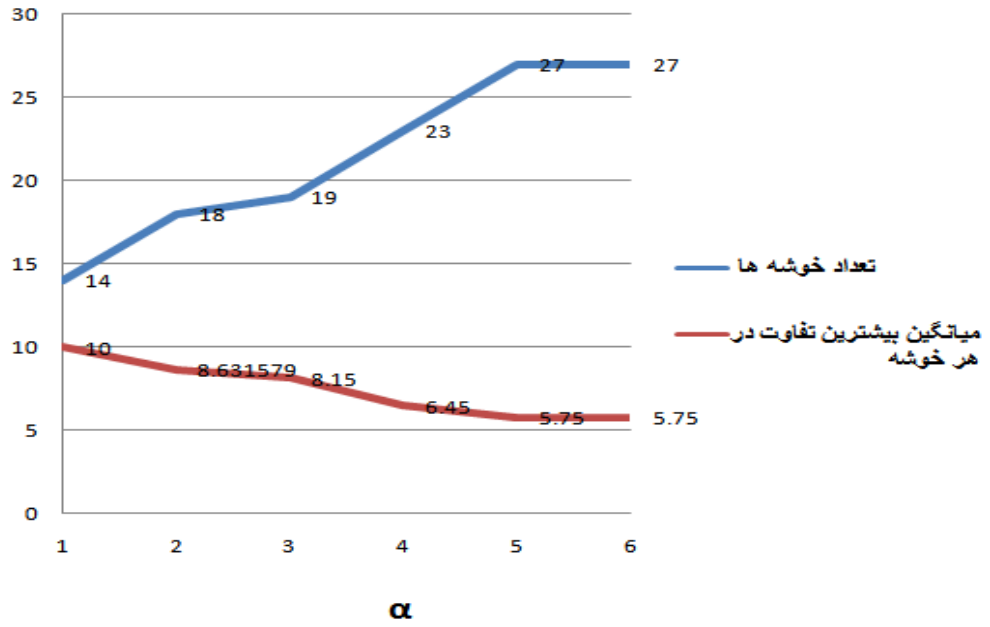
شکل ۴-۲: شبه‌کد واحد کاری دوم جهت استخراج الگو

گام دوم) اعمال الگوریتم(تنظیم پارامترهای فرایند خوشه‌بندی)

با توجه به فرمول (۱-۳) از الگوریتم خوشه‌بندی ارائه شده، نیاز به تنظیم دو پارامتر اولیه α و β جهت خوشه‌بندی مشابه‌ترین موجودیت‌ها داریم. به ازای این دو پارامتر نتیجه‌ی خوشه‌بندی از دو جنبه مقایسه می‌شود. جنبه اول، تعداد خوشه‌های ایجاد شده‌است که درواقع نشان‌دهنده‌ی تعداد جداول HBase است. جنبه دوم، میانگین بیشترین تفاوت‌های بین الگوهای درون یک خوشه نسبت به نماینده‌ی خوشه را نشان می‌دهد. منظور از تفاوت، تعداد صفاتی در نماینده‌ی خوشه است که در الگوی مورد نظر وجود ندارد. هدف از الگوریتم پیشنهادی در درجه‌ی اول کم کردن میزان تفاوت بین الگوهای رخ داده برای موجودیت‌ها است که اگر این محاسبه‌ی تفاوت، با نماینده‌ی خوشه انجام شود

کافی است. درست است که در HBase سربار *NULL* وجود ندارد، اما هرچه تشابه موجودیت‌ها در یک جدول بیشتر باشد، فضای مورد جست‌وجو کاهش یافته و هزینه‌ی جست‌وجو در جدول مورد نظر برای الگوهایی از موجودیت‌ها کاهش می‌یابد. علت این امر این است که HBase به بلاک‌های کمتری برای بازیابی موجودیت‌های دقیق رجوع می‌کند. با این اولویت ذکر شده، ابتدا کمترین β که به ازای آن اکثر الگوها با مقادیر مختلف α خوشه‌بندی می‌شوند، یافت شده و سپس به ازای β بدست آمده، مقادیر مختلف α الگوریتم، آزمون شده تا بهترین مقدار α یافت شود.

با توجه به مجموعه‌داده‌های مورد استفاده، مقدار β که به ازای آن داده‌ها در حالت خوشه‌بندی شده قرار می‌گیرند، در آزمایشات برابر ۳,۵ است. تعداد خوشه‌ها و میانگین تفاوت الگوها به ازای مقادیر مختلف α با مقدار ثابت β=۳,۵ در شکل(۴-۳) و برای مجموعه‌داده‌ی DBLP نشان داده‌شده-است.



شکل ۴-۳: نسبت تغیرات تعداد خوشه‌ها و میانگین تفاوت در خوشه‌ها با توجه به مقادیر مختلف α و ثابت β=۳,۵

با توجه به نمودار، به ازای مقدار α = ۵ تفاوت بین اعضای خوشه و نماینده‌ی خود در کمترین مقدار خود قرار داشته و از α = ۷ به بالاتر، چون الگوهای خوشه‌بندی نشده تولید می‌شود در شکل

نیز نشان داده نشده‌اند. نظر به اینکه اولویت اول برای انتخاب α بهینه، کم بودن میزان تفاوت خوشه‌ها است، بنابراین $\alpha = ۵$ تنظیم می‌شود. با توجه به امکان مخزن اتصالات HBase به جداولِ موجود در گره‌های توزیعی، هزینه دسترسی به جداول پایین بوده و مقدار ۲۷ مقدار قابل قبولی است[Med11]. نهایتا گام سوم الگوریتم یعنی ایجاد نماینده‌ی خوشه‌ها اجرا می‌شود.

در ضمن علی‌رغم مرحله‌ی استخراج الگوها که به صورت توزیعی انجام شده است، خوشه‌بندی روی داده‌ها به صورت متمرکز انجام می‌شود؛ چرا که حجم الگوهای استخراج شده از مجموعه‌داده‌ی انتخابی، بسیار کم و در حد کیلوبایت است و در حجم‌های بالاتر می‌توان از روش‌های یادگیری استفاده نمود.

۲-۱-۱-۴- ایجاد شِمای جداول در HBase

HBase در هنگام ایجاد جداول امکانات زیادی را به کاربر ارائه می‌دهد تا با توجه به ذات داده‌ها این امکانات روی جداول تنظیم شوند. این امکانات می‌تواند روی سرعت پردازش داده‌ها، تاثیر بالایی داشته باشد.

اولین مورد تنظیم BloomFilters است. همانطور که گفته شد، با استفاده از این امکان قبل از باز کردن StoreFileها، بلاک BloomFilters به حافظه آورده شده و سیستم از وجود و یا عدم وجود یک سطر به سرعت با خبر می‌شود. بنابراین ممکن است در این بین یک سری از StorFileهای غیر لازم جست‌وجو نشوند. در تنظیمات مربوطه، BloomFilters در سطح سطر انتخاب شده‌است. البته حضور BloomFilters خود باعث اشغال حافظه می‌شود که باید در هنگام در نظر گرفتن فضای حافظه به این نکته دقت شود.

دومین مورد تنظیم اندازه‌ی بلاک‌های هر StoreFile است. این مقدار برابر مقدار پیش‌فرض HBase یعنی ۶۴ کیلوبایت در نظر گرفته می‌شود.

سومین مورد، تنظیم کش بلاک‌ها است. برای هر جدول می‌توان تنظیم نمود که شاخص بلاک‌ها در درون سرورهای ناحیه‌ای کش شوند. تنظیم این مورد بسیار مهم است چرا که در غیر این صورت سرور ناحیه‌ای مداوم درگیر خواندن داده‌ها از حافظه می‌شود که سرعت را بسیار پایین می-آورد. این مورد نیز برای پیاده‌سازی سیستم موردنظر تنظیم شده‌است.

چهارمین مورد، تنظیم تعداد نسخه‌های داده‌های یک سلول است. با توجه به اینکه از امکان نسخه‌بندی، برای ذخیره‌ی صفت‌های چند مقداری استفاده‌شده‌است، این مقدار برای هر ستون برابر بزرگترین مقدار به دست‌آمده یعنی ۶۰۰ در نظر گرفته شده‌است.

پنجمین مورد، استفاده از امکان جداسازی جداول در HBase و یا تعریف یک مکانیزم جداسازی و تقسیم جداول بین سرورهای ناحیه‌ای است. خود HBase امکان جداسازی خودکار جداول را می‌دهد بنابراین در حال حاضر از این امکان استفاده‌شده‌است.

۲-۱-۴- پیاده‌سازی بخش ذخیره‌سازی داده‌ها

در این بخش موجودیت‌ها از درون مجموعه‌داده استخراج و سپس در جدول مربوطه، ذخیره می‌شوند. با توجه به اینکه ممکن است یک موجودیت در اسناد مختلفی توصیف شده‌باشد، از این رو باید پردازشی روی فایل‌ها انجام گیرد تا اطلاعات هر موجودیت همراه با او ذخیره شود. برای پردازش فایلها از مدل برنامه‌نویسی نگاشت-کاهش استفاده شده‌است. شبه‌کد این واحد کاری در شکل(۴-۴) قابل مشاهده‌است. در تابع نگاشت ابتدا موجودیت مورد نظر استخراج و سپس آن موجودیت به عنوان کلید و سه‌گانه‌ی مورد نظر به عنوان مقدار به بخش کاهش داده می‌شود.

در مرحله‌ی نگاشت از سه‌گانه‌ی مورد نظر، موجودیت استخراج می‌شود. سپس موجودیت به عنوان کلید و سه‌گانه مورد نظر به عنوان مقدار در خروجی چاپ می‌شوند. در مرحله‌ی کاهش به‌ازای کلید مورد نظر یعنی موجودیت‌ها، سه‌گانه‌های مربوط به آن موجودیت در کل مجموعه‌داده در

دسترس است. بنابراین مستقیم سه‌گانه‌ها در خروجی چاپ می‌شوند. در واقع با این واحد کاری یک فایل از سه‌گانه‌ها که براساس موجودیت‌ها مرتب شده، داریم.

```
1: map(key,value)
2: //key:irrelevant
3: //value: triples in N-triple Format
4: Subject=getSubject(value)
5: Context.Write(Subject,value)

1: Reduce(Key,iterator Values)
2: //value: triples in N-triple Format
3: for(value in Values)
6:         Context.Write(value,null)
```

شکل ۴-۴: شبه کد مرتب نمودن مجموعه داده بر اساس Subject

برای ذخیره‌ی داده‌ها، HBase امکانات مختلفی را ارائه داده است. با توجه به تعداد جداولی که در روش پیشنهادی ایجاد می‌شود بنابراین مناسب‌ترین گزینه، روش ارائه شده توسط خود HBase یعنی PutList است. در روش پیشنهادی، داده‌ها از فایل خروجیِ واحد کاری سوم خوانده می‌شوند. هنگامی‌که یک موجودیت و صفات و مقدار آن کشف شد، بر اساس الگوی صفاتش بهترین نماینده‌ی خوشه یافت می‌شود. سپس بعد از ساختن کلید سطر، با توجه به نماینده‌ی خوشه، نام جدول HBase مورد نظر بازیابی می‌شود. اما در این مرحله داده‌ها مستقیم در جدول ذخیره نمی‌شوند. بلکه در این روش ابتدا یک نمونه از کلاس Put که در آن کلید سطر و ستون‌ها و مقدار ستون‌ها تنظیم می‌شوند، ساخته می‌شود و براساس موجودیت مورد نظر کلید سطر و ستون‌ها و مقدار هر ستون جایگذاری می‌شود. سپس این موجودیت در یک لیست که مخصوص جدولی است که باید در آن ذخیره شود، نگه‌داری می‌شود. حال اگر تعداد موجودیت‌های این لیست به ازای جدول مورد نظر به یک مقدار ثابت مانند ۳۰ رسید، لیست این موجودیت‌ها یک‌جا به جدول مورد نظر HBase جهت ذخیره‌سازی فرستاده می‌شوند. این‌کار دو مزیت دارد؛ اولا به ازای فقط یک موجودیت یک درخواست به سرور اصلی و سپس سرورهای ناحیه‌ای فرستاده نمی‌شود. دوما با این‌کار HBase لیست Putها را براساس

محدوده‌ی کلیدسطر مورد نظر مرتب کرده و به طور جداگانه به هر سرورناحیه‌ای که آن ناحیه را پشتیبانی می‌کند مجموعه Putها را می‌فرستد. بنابراین در یک اتصال به سرورناحیه‌ای حجم زیادی از داده را می‌فرستد.

در پیاده‌سازی روش پیشنهادی این مقدار ثابت همان ۳۰ در نظر گرفته‌شده‌است.

۳-۱-۴- پیاده‌سازی بخش بازیابی داده‌ها

امکانات کلی HBase که در بازیابی داده‌ها تاثیرگذار خواهد بود عبارت‌اند از :

۱) استفاده از صافی‌ها

۲) مخزن اتصالات HBase

۳) استفاده از کشینگ HBase – سیستم مشتری در هنگام بازیابی داده‌ها، برای بازیابی هر سطر به سرورناحیه‌ای متصل می‌شود. بنابراین برای افزایش سرعت، مقدار آن ۵۰۰ تنظیم شده‌است. بنابراین در یک اتصال به سرورناحیه‌ای مورد نظر، ۵۰۰ سطر یک‌جا بازیابی می‌شود.

۴) استفاده از BloomFilterها که در هنگام ایجاد جداول، تنظیم شده‌اند.

همچنین با توجه به اینکه ممکن است هنگام بازیابی داده‌ها بر اساس الگوی درخواستی در پرس‌وجو به بیش از یک جدول دسترسی صورت گیرد بنابراین از امکان چند نخی استفاده شده‌است. به این نحو که در یک حلقه به ازای هر نماینده‌ی خوشه که الگوی مورد نظر کاربر را حمایت کرد، جهت جست‌وجوی پشوندی صافی‌ها ساخته، و نام جدول بازیابی می‌شود. سپس نام جدول HBase و صافی‌ها به یک نخ داده‌شده تا اطلاعات را بازیابی و نمایش دهد.

۴-۲- ارزیابی

سیستم پیشنهادی در هر سه بخشِ پیاده‌سازی شده مورد ارزیابی قرار می‌گیرد. ابتدا در بخش(۴-۲-۱)مجموعه‌داده‌ی انتخابی معرفی می‌شود. معیارهای ارزیابی را در بخش (۴-۲-۲) بیان می‌کنیم. سپس در بخش(۴-۲-۳) آخرین کار مشابه روی جداول رابطه‌ای مورد ارزیابی اولیه قرار می‌گیرد تا مشخص شود علت ارائه الگوریتم جدید خوشه‌بندی موجودیت به چه علتی بوده‌است. سپس در بخش (۴-۴-۲) به ارزیابی سیستم فعلی با آخرین شِمای مشابه ارائه شده در جداول HBase می‌پردازیم.

۴-۲-۱- مجموعه‌داده ی انتخابی

در روش پیشنهادی از دو مجموعه‌داده‌ی LUBM [lubm] با حجم ۵ گیگابایت و DBLP [dblp] با حجم ۲,۵ گیگابایت استفاده شده‌است. LUBM در واقع یک تولیدکننده‌ی داده در حوزه‌ی دانشگاه است که با تولید تعداد دانشگاه‌های مختلف حجم‌های مختلفی از داده‌ها را تولید می‌کند. DBLP نیز مجموعه‌داده‌ای مربوط به مقالات در حوزه علوم کامپیوتر است.

۴-۲-۲- معیارهای ارزیابی

با توجه به اجزای سیستم پیشنهادی، ارزیابی از سه جنبه‌ی استخراج ساختار، ذخیره‌سازی و بازیابی انجام می‌گیرد.

۴-۲-۲-۱- معیارهای ارزیابی بخش استخراج ساختار

سرعت ایجاد شِما

با استفاده از این معیار بررسی می‌شود که هر سیستم چه عملیاتی را برای استخراج ساختار از داده‌ها و سپس ایجاد شِما انجام می‌دهد. با توجه به این که روش‌پیشنهادی از خوشه‌بندی نیز استفاده

می‌کند، الگوریتم مورد نظر نیز مورد ارزیابی قرار می‌گیرد. بنابراین در این بخش سیستم‌ها از نظر الگوریتم‌ها و پیش‌پردازش‌های لازم روی داده‌ها مورد ارزیابی قرار می‌گیرند.

۲-۲-۲-۴- معیارهای ارزیابی بخش ذخیره‌سازی

در این بخش سیستم‌ها از دو جنبه‌ی اندازه شاخص و زمان بارگذاری شاخص مورد ارزیابی قرار می‌گیرند.

اندازه ی شاخص(سربارهای تحمیلی به سیستم در طراحی شِما)

از جمله عواملی که می‌تواند عملکرد یک شِما را تحت تاثیر قرار دهد، اندازه شاخص خواهد بود. در طراحی یک سری از شِماها افزونگی داده‌ها به حدی می‌رسد که اندازه شاخص از حد معقول خود بسیار بالاتر می‌شود. بنابراین اگر در این مورد درست اندیشیده نشود در برابر حجم داده‌های موجود در جهان وب، اندازه شاخص‌ها بیش از توان سیستم در کنترل آن خواهد بود. البته درست است که در محیط توزیع شده شاید تاحدی این معیار از سطح اهمیت خود پایین‌تر قرار داشته‌باشد، اما باید درنظر داشت که هنگامی اندازه شاخص در داده RDF از حد معقول خود بالاتر رود به معنای حضور افزونگی روی داده‌هاست و این امر می‌تواند به عنوان مشکل بزرگی در یک سری از وضعیت‌ها همچون بروزرسانی داده‌ها باشد[Tre12][Del10].

زمان بارگذاری شاخص مربوطه(سرعت ایجاد شاخص)

معیار بارگذاری شاخص در واقع اشاره به زمان شاخص‌گذاری داده‌ها دارد. زمان بارگذاری در واقع زمانی است که سیستم به صورت برون‌خطی[۱] در حال شاخص‌گذاری داده‌ها است. اما اگر همین

---

[۱]- Offline

سیستم به نحوی کند باشد که غیر قابل چشم پوشی باشد تمام عملکرد یک سیستم بازیابی اطلاعات همچون موتور جست‌وجو را زیر سوال خواهد برد[Neu10].

۳-۲-۲-۴- معیارهای ارزیابی بخش بازیابی

این بخش از بازیابی کلیدی‌ترین بخش است. مهمترین معیاری که نمی‌توان از آن چشم پوشی کرد، آزمون سرعت پاسخگویی سیستم در برابر پرس‌وجوها است. در واقع ابتدا یک سری از پرس-وجوها متناسب با مجموعه‌داده مورد نظر، آماده می‌شود. سپس این پرس‌وجوها روی سیستم مورد نظر اعمال می‌شود و زمان اجرای آنها ثبت می‌شود.

۳-۲-۴- ارزیابی اولیه

این بخش به ارزیابی سیستمی‌تخصیص دارد که از نظر شِما تاحدودی مشابه روش پیشنهادی بوده و برای پایگاه داده‌های رابطه‌ای ارائه شده‌است[Lev09]. سیستم مورد نظر، به طور مفصل در بخش (۲-۲) شرح داده‌شده‌است.

۱-۳-۲-۴- مشخصات محیط اجرایی سیستم

ارزیابی روی سیستمی۷ هسته‌ای با حافظه‌ی اصلی ۱۶گیگابایت انجام شده‌است. مجموعه‌داده‌ی انتخابی نیز حدود ۷۰۰,۰۰۰ سه‌گانه از مجموعه‌داده DBLP است.

۲-۳-۲-۴- ارزیابی در بخش استخراج ساختار و شاخص‌گذاری

نتایج حاصل از ارزیابی در جدول (۱-۴) نشان داده‌شده‌است. در هنگام اعمال الگوریتم، روش مورد مقایسه برابر ۱ ثانیه عمل می‌کند. الگوریتم سیستم مورد مقایسه نتایج خوبی روی داده‌های کم و حدود ۱۰ مگابایت دارد[Lev09]. اما همانطور که مشهود است زمانی که حجم داده‌ها به تقریبا ۱۰۰ مگابایت می‌رسد، شِمای ارائه شده که مبتنی بر موجودیت است در خروجی به شِمای مبتنی بر بخش‌بندی عمودی شبیه‌تر شده‌است. بدین معنا که ۱۲۷ تعداد از جداول، جداول مبتنی بر بخش-

بندی عمودی‌اند. یعنی جداولی هستند که نامشان predicate مورد نظر و دارای دو ستون برای subject و object است. البته همانطور که از آمار پیداست تعداد ستون‌های جدول ویژگی، نیز بسیار کم است. علت خروجی این الگوریتم وجود دو پارامتر درصد NULL و درصد حمایت است. در واقع زمانی که حجم داده‌ها زیاد می‌شود پراکندگی داده‌ها نیز زیاد می‌شود. همچنین به خاطر افزایش الگوها و تنوع در آنها درصد حمایت نیز برای یک الگو بسیار کاهش می‌یابد. بنابراین سیستم نمی‌تواند دسته‌بندی روی الگوها را به درستی انجام دهد. به همین دلیل روش مورد نظر به تولید جداول دودویی بیشتری می‌پردازد. با این تفاسیر هدف ما در این تحقیق ارائه یک روش دیگر برای خوشه-بندی موجودیت‌ها با در نظر گرفتن هم‌پوشانی و استفاده از جداول HBase است.

جدول ۴-۱: اندازه ی شاخص

شِمای جداول	تعداد جداول چندتایی	میانگین تعداد ستون هر جدول	تعداد جداول دودویی
الگوریتم مبتنی بر جدول رابطه‌ای	۸(بدون همپوشانی)	۲,۵	۱۲۷

۴-۲-۴- ارزیابی سیستم پیشنهادی

در این بخش سیستم پیشنهادی با شِمایی مشابه با آخرین شِمای ارائه شده یعنی شِمای مبتنی بر تک جدولی [TAK]، مورد مقایسه و ارزیابی قرار می‌گیرد. همانطور که در بخش (۲-۱-۲-۱) گفته شد، این شِما تمام موجودیت‌ها را در سطرهای یک جدول قرار می‌دهد به نحوی که صفات آن موجودیت در ستون جدول قرار می‌گیرد.

۱-۴-۲-۴- مشخصات محیط اجرایی سیستم

در سیستم پیشنهادی از HBase۰,۹۴,۱۱ و Hadoop۱,۲,۱ استفاده‌شده‌است. با توجه به معماری HBase، سیستم پیشنهادی توزیع شده بوده و دارای ۵ گره به عنوان سرورهای ناحیه‌ای[۱] و یک گره به عنوان گره‌ی اصلی[۲] Hadoop و HBase است. هر شش گره، هشت‌هسته‌ای بوده و دارای ۸ گیگابایت حافظه اصلی هستند.

۲-۴-۲-۴- ارزیابی در بخش استخراج ساختار

شِمای تک جدولی هیچگونه عملیاتی را جهت استخراج ساختار انجام نمی‌دهد بلکه بعد از تشخیص یک موجودیت، مستقیم داده‌ها را در یک جدول می‌ریزد. بنابراین نتایج جدول(۲-۴) مربوط به الگوریتم خوشه‌بندی در روش پیشنهادی است.

جدول ۴-۲: نتایج عملیات استخراج ساختار از داده‌ها در روش پیشنهادی(به ثانیه)

عملیات استخراج ساختار	زمان شناسایی و استخراج الگوها	زمان اعمال الگوریتم خوشه-بندی	زمان ایجاد شِما
مجموعه‌داده DBLP	۱۰۵	۲	۶۰
مجموعه‌داده LUBM	۹۴۳	۰,۱	۸

با توجه به الگوریتم اعمالی، آمارهای به دست آمده از ساختار مجموعه‌داده‌ها در جدول(۳-۴) نمایش داده‌شده‌است. دیدگاهی که می‌توان از این دو مجموعه‌داده کسب کرد این است که داده‌ها در مجموعه‌داده DBLP نسبت به مجموعه‌داده LUBM پراکندگی بیشتری دارند.

---

[8] Region Server

[۲] Master Node

جدول ۴-۳ :آمارهای حاصل از پردازش مجموعه داده‌ها در الگوریتم پیشنهادی

مجموعه‌داده/ وضعیت داده‌ها	DBLP	LUBM
تعداد سه‌گانه‌ها	۴۹۴۰۰۳۰	۲۶۷۱۱۸۱۱
تعداد موجودیت‌ها	۲۵۴۸۶۵۲	۴۲۱۴۵۰۱
تعداد جداول(تعداد خوشه‌ها)	۲۷	۴
میانگین طول ستون‌های هر جدول (تعداد صفت‌ها)	۲۰,۶۷۸۵۷۲	۵,۴
نرخ NULL (میانگین بیشترین NULL در هر خوشه)	۵,۷۵	۱,۸
بیشترین تفاوت نسبت به نماینده‌ی خوشه	۱۷	۳

۳-۴-۲-۴- ارزیابی در بخش ذخیره‌سازی

ارزیابی‌های انجام شده روی هر مجموعه‌داده به تفکیک در جدول (۴-۴) و (۴-۵) نمایش داده-شده‌است. با توجه به اینکه شاخص‌گذاری به صورت برون خطی انجام می‌گیرد، بنابراین تفاوت اندک این دو روش در ذخیره‌سازی داده‌ها قابل چشم‌پوشی است. دقت گردد که در بخش استخراج از ساختار روش‌پیشنهادی هزینه‌ی اضافی اجرای الگوریتم خوشه‌بندی را دارد که شِمای تک جدولی این هزینه را ندارد.

جدول ۴-۴: ارزیابی شِمای بیشنهادی و شِمای تک جدولی از نظر شاخص‌گذاری در مجموعه‌داده DBLP

سیستم‌ها	پردازش مجموعه-داده	استخراج موجودیت‌ها	ذخیره‌سازی موجودیت‌ها	نرخ ذخیره‌سازی موجودیت‌ها	نرخ ذخیره-سازی سه‌گانه
سیستم پیشنهادی	۲دقیقه و ۴۵ ثانیه	۴ دقیقه و یک ثانیه	۱۵دقیقه	۲۸۳۱,۸ موجودیت/ثانیه	۱۶۶۰۰,۳ سه‌گانه/ثانیه
شِمای تک جدولی	۲دقیقه و ۴۵ ثانیه	۴ دقیقه و یک ثانیه	۱۸ دقیقه و ۲۶ ثانیه	۲۳۰۴,۳ موجودیت/ ثانیه	۱۳۵۰۸,۱۶ سه‌گانه/ثانیه

جدول ۴-۵: ارزیابی شِمای بیشنهادی و شِمای تک جدولی از نظر شاخص‌گذاری در مجموعه‌داده LUBM

سیستم‌ها	پردازش مجموعه-داده	استخراج موجودیت‌ها	ذخیره‌سازی موجودیت‌ها	نرخ ذخیره‌سازی موجودیت‌ها	نرخ ذخیره-سازی سه‌گانه
سیستم پیشنهادی	۱۶ دقیقه	۲ دقیقه و دو ثانیه	۲۲دقیقه ۳۴ ثانیه	۳۱۱۲,۶ موجودیت/ثانیه	۱۹۷۲۸,۰۷ سه‌گانه/ثانیه
شِمای تک جدولی	۱۶ دقیقه	۲ دقیقه و دو ثانیه	۲۳ دقیقه و ۳۲ ثانیه	۲۹۸۰ موجودیت/ ثانیه	۱۸۸۹۰,۹ سه‌گانه/ثانیه

۴-۴-۲-۴- ارزیابی در بخش بازیابی

برای ارزیابی سیستم‌ها ابتدا یک سری از پرس‌وجوها که تمام الگوهای دسترسی را پوشش می‌دهند روی مجموعه‌داده‌ی DBLP تعریف می‌شود. همچنین مجموعه‌داده‌ی LUBM نیز دارای ۱۴ پرس‌وجوی استاندارد است. در بین این چهارده پرس‌وجو، ۸ پرس‌وجو مبتنی بر موجودیت هستند. با توجه به اینکه در این نوع پرس‌وجوها استدلال روی مجموعه‌داده نیز در نظر گرفته شده‌است؛ بنابراین تنها سه‌پرس‌وجو را می‌توان برای ارزیابی فعلی سیستم پیشنهادی استفاده نمود.

هر سه‌گانه را در قالب SPO می‌توان نشان داد که طبیعتا هر الگوی دسترسی از ۸ حالت خارج نیست. بدین معنی که در هر الگوی دسترسی ممکن است یکی از اعضای سه‌گانه متغیر باشد که آن را با V نمایش می‌دهیم و یا ثابت باشد که آن را با C نشان می‌دهیم. در این بین ۲ الگوی پرس‌وجوی VVV و CCC وجود دارند که به علت پشتیبانی محدود کاربران، در ارزیابی فعلی مورد استفاده قرار نمی‌گیرد. بنابراین ۶ حالت برای اعمال پرس‌وجو به سیستم وجود دارد. بر مبنای این شش دسترسی مختلف، پرس‌وجوهای مختلف تعریف شده‌است که در جدول (۴-۶) آورده شده‌اند.

مفهوم گزینش‌پذیری در ادامه شرح داده‌می‌شود. هرچه گزینش‌پذیری پایین باشد به این معنی است که پرس‌وجوی اعمالی دارای متغیرهای (V)زیادی روی سه‌گانه‌ی مورد پرسش است. بنابراین برای اجرای پرس‌وجو، سیستم با داده‌های بیشتری سروکار دارد که از سرعت اجرای پرس‌وجو می‌کاهد. بنابراین بر این اساس می‌توان پرس‌وجوها را دسته بندی نمود. در این دسته‌بندی اگر پرس‌وجویی تنها یک V داشت، آن پرس‌وجو دارای گزینش پذیری بالایی است و بسته به حجم داده باید سریعتر به پاسخ برسد. نتایج حاصل از اجرای پرس‌وجوها در هر دو مجموعه‌داده در شکل(۴-۵) و (۴-۶) نشان داده شده‌است.

جدول ۴-۶: پرس‌وجوهای مورد آزمایش و ویژگی های هریک

پرس‌وجوی LUBM	پرس‌وجوی DBLP	‘گزینش پذیری	تعداد صفات متغیر	تعداد سه‌گانه‌های مورد پرسش	الگوهای دسترسی
LQ1,LQ3	DQ1	بالا	-	1	VCC
–	DQ2	بالا	-	1	CCV
–	partial	-	-	-	CVC
–	partial	-	-	-	CVV
LQ4	DQ3	پایین	-	3	VCV
–	DQ4	پایین	-	3	VCV
–	DQ5	پایین	-	10	VCV
–	DQ6	پایین	-	5	VCV
–	DQ7	پایین	-	1	VCV
–	partial DQ8	پایین	2	4	VVC
–	DQ9	پایین	-	3	optional
–	DQ10	بالا	-	1	MultiValue Predicate (CCV)

نتایج پرس‌وجوهای اجرایی روی مجموعه‌داده‌ی DBLP

شکل ۴-۵: نتایج پرس‌وجوهای اجرایی روی مجموعه‌داده‌ی DBLP

جهت ارائه یک دیدگاه کلی‌تر، میزان نتایج بازگشتی و تعداد جداولی که در روش پیشنهادی مورد دسترسی قرار می‌گیرند، در جدول(۷-۴) نمایش داده‌شده‌است.

جدول ۴-۷: نتایج حاصل از پرس‌وجو های اعمالی در مجموعه داده DBLP

پرس‌وجوها	DQ1	DQ2	DQ3	DQ4	DQ5
تعداد نتایج	1	1	7133	965074	192
تعدادجداول	7	27	4	13	1
پرس‌وجوها	DQ6	DQ7	DQ8	DQ9	DQ10
تعداد نتایج	348968	2548478	1	382278	313
تعدادجداول	3	27	14	6	4

تحلیل نتایج اجرای پرس‌وجوهای مجموعه‌داده DBLP

DQ1) در این پرس‌وجو، هدف یافتن مقاله‌ای است که دارای یک شناسه‌خاص باشد.

هدف اصلی روش پیشنهادی در اجرای پرس‌وجوها، ایجاد یک صافی مبتنی بر پیشوند است. بنابراین در اجرای هر پرس‌وجو به دنبال ساختن این چنین پیشوندهایی هستیم.

همانطور که گفته شد ساختار یک کلید سطر برای موجودیت A با الگوی صفات p به شکل زیر تعریف می‌شود:

K=<شناسه‌ی نماینده‌ی (p)+شناسه‌ی (p)> + A

بنابراین هنگامی که این پرس‌وجو روی سیستم پیشنهادی اجرا می‌شود؛ ابتدا در بین نمایندگان خوشه‌ها جست‌وجو می‌شود تا نمایندگانی که الگوی این پرس‌وجو را در بین صفات عضو خود حمایت می‌کنند، بازیابی شوند.

بعد از بدست آوردن شناسه‌ی نماینده‌ی خوشه، جهت به دست آوردن شناسه‌ی الگوهایی که مطابق با الگوی مورد پرس‌وجو هستند، درون خوشه‌ی نماینده‌ی مورد نظرنیز جست‌وجو می‌شود تا شناسه‌ای از اعضا که مطابق با الگوی مورد پرس‌وجو هستند استخراج شوند. در نهایت با الحاق این دو

شناسه، یک صافی مبتنی بر پیشوند را می‌توان روی داده‌ها اجرا کرد. سپس با استفاده از فراداده‌ها، نام جداولی که از این ساختار داده پشتیبانی می‌کنند بازیابی و به جداول مورد نظر دسترسی صورت می‌گیرد.

اجرای این پرس‌وجو در شِمای تک جدولی منجر به ایجاد یک صافی مبتنی بر ستون و مقدار و سپس اجرای آن روی یک جدول می‌شود.

همانطور که در جدول (۷-۴) مشاهده می‌شود هنگام اجرای این پرس‌وجو، ۷ نماینده‌ی خوشه و الگوهای عضو آن خوشه، از پرس‌وجو حمایت می‌کنند و بنابراین باید به ۷ جدول دسترسی صورت گیرد. اما در شِمای تک جدولی تنها به یک جدول دسترسی صورت می‌گیرد. اما باید گفت با استفاده از مخزن اتصالات HBase، هزینه دسترسی به چند جدول برابر هزینه دسترسی به یک جدول است که هردو روش این هزینه را دارند[Med11]. بنابراین این دسترسی‌ها به جداول، تاثیر منفی در عملکرد سیستم پیشنهادی ندارد.

علت اینکه هزینه‌ی اجرای این دو پرس‌وجو روی هردو سیستم یکی است این است که به نظر می‌آید برای الگوهای دسترسی با یک صفت، هردو صافی عملکرد یکسانی را از خود به نمایش گذاشته‌اند. در صورتی که انتظار داشتیم صافی مبتنی بر پیشوند به دلیل اینکه در کلید سطر قرار دارد، سرعت اجرای بالاتری را داشته باشد. این نشان‌دهنده‌ی این است که صافی مبتنی بر ستون و مقدار HBase هنگام بازیابی داده‌ها با استفاده از یک ستون، عملکرد قابل قبولی را از خود نشان می-دهد.

DQ2) در این پرس‌وجو، هدف یافتن نوع یک موجودیت مقاله از نظر پذیرفته شدن در کنفرانس است.

نحوه‌ی اجرای این پرس‌وجو در سیستم پیشنهادی در Q1) شرح داده‌شده‌است. اجرای این پرس‌وجو در هر دوسیستم یکسان و مشابه‌است. سیستم پیشنهادی با بدست آوردن صافی‌های مختلف پیشوندی از نمایندگان و اعضای خوشه‌ها و الحاق آن با موجودیت مورد جست‌وجو، پرس‌وجو را پاسخ می‌دهد. شِمای تک جدولی با استفاده از صافی مبتنی بر ستون و صافی مبتنی بر سطر این‌کار را انجام می‌دهد. بنابراین اجرای این دو پرس‌وجو طبیعتا یکسان است.

DQ3) در این پرس‌وجو هدف بازیابی یک مقاله با ۳ صفت است..

نحوه‌ی اجرای این پرس‌وجو در سیستم پیشنهادی در Q1) شرح داده‌شده‌است. شِمای تک جدولی نیز از سه صافی مبتنی بر ستون را روی جدول خود اعمال می‌کند. همانطور که انتظار می‌رود سیستم پیشنهادی با استفاده از صافی پیشوندی و تنها رجوع به جداولی که این الگوها را پشتیبانی می‌کنند از شِمای تک جدولی بهتر عمل می‌کند.

DQ4) در این پرس‌وجو هدف یافتن مقالاتی است که دارای سه صفت باشند.

اجرای این پرس‌وجو در دو سیستم مشابه Q3) است. با این تفاوت که حجم داده‌های بازگشتی بسیار بالاتر است. باید گفت در سیستم‌های محاسبه ابری از اجرای پرس‌وجوهایی که منجر به بار زیاد روی سرورها شود جلوگیری می‌شود. به عنوان مثال گوگل تنها حجم محدودی از داده‌ها رابازیابی می‌کند تا مشکل بار اضافی[۱] پیش نیاید. در این تحقیق برای گزارش‌گیری، تمام داده‌های قابل قبول بازگردانده می‌شوند. همانطور که مشاهده می‌شود، حجم داده‌ی بازگردانده شده ، علت بالا رفتن زمان پرس‌وجو را برای هر دو سیستم نشان می‌دهد.

---

[۱]- Over Loading

DQ5) هدف از اجرای این پرس‌وجو بازیابی یک موجودیت با ۱۰ صفت است.

اجرای این پرس‌وجو در هر دو سیستم مشابه پرس‌وجوی Q1) است. این پرس‌وجو نشان می-دهد که هرچقدر یک موجودیت با صفات بیشتری مورد پرس‌وجو قرار گیرد و یا به عبارتی دقیق‌تر بیان شود، تعداد جداول مورد دسترسی کمتر بوده، و می‌توان سریعتر پاسخ پرس‌وجوها را یافت. همچنین اجرای این پرس‌وجو در شِمای تک جدولی نشان می‌دهد که هر قدر تعداد صفات مورد درخواست بیشتر باشد هزینه‌ی جست‌وجو نیز کمتر خواهد بود.

DQ6)هدف از اجرای این پرس‌وجو بازیابی یک موجودیت با یک سری از صفات است.

نحوه‌ی اجرای این پرس‌وجو همانند اجرای پرس‌وجوهای دیگری است که تا به الان گفته شده-است. همانطور که در جدول (۴-۷)مشاهده می‌شود، زیاد شدن حجم داده‌ی موردنظر زمان اجرای پرس‌وجوها را بالا می‌برد.

DQ7) هدف از این پرس‌وجو یافتن نوع همه‌ی موجودیت‌ها است.

این نوع پرس‌وجو سیستم را متحمل بازیابی یک حجم زیاد از داده می‌کند و هدف آن آزمون همین نرخ بازیابی داده‌ها بدون جست‌وجوی پیچیده‌است. در این پرس‌وجو به نظر می‌آید که سیستم پیشنهادی به خاطر تخصیص یک نخ به جدول مورد پرس‌وجو و سپس جمع آوری و نمایش داداه‌ها، از سیستم مبتنی بر تک جدولی بهتر عمل می‌کند.

DQ8) هدف از این پرس‌وجو یافتن یک موجودیت با دو صفت صریح و دو صفت غیر ذکر شده(متغیر) است که برای آن دو صفت متغیر، تنها مقدارشان در پرس‌وجو آمده‌است.

همانطور که از پیاده‌سازی سیستم پیشنهادی روشن است؛ این سیستم بعد از پردازش پرس‌وجو و استخراج الگوها از آن، به خوشه‌هایی که از آن الگو حمایت می‌کنند رجوع می‌کند. اما پرس‌وجو-هایی را در نظر بگیرید که در آنها predicate به عنوان متغیر باشد. به عنوان مثال در الگوهای

دسترسی که در جدول(۶-۴) مشاهده می‌شود الگوهایCVV و VVC و CVC دارای صفات متغیر است. در این نوع پرس‌وجوها که درآنها الگو قابل استخراج نیست در هر دو سیستم قابل پاسخگویی کامل وجود ندارد. چرا که در شِمای تک جدولی نیز برای ساخت صافی ستون نیاز به دانستن صفت مورد نظر است. اما در روش پیشنهادی این نوع پرس‌وجوها را می‌توان به صورت جزئی پاسخ داد. منظور ار جزئی این است که اگر کاربر در پرس‌وجوی خود حداقل یک صفت را مشخص کند، سیستم پیشنهادی به همه‌ی خوشه‌ها مراجعه کرده و به دنبال نمایندگانی می‌گردد که همان یک صفت را حمایت کنند و از طرفی تعداد اعضای آن نماینده‌ی خوشه(مجموعه صفاتی که آن نماینده حمایت می‌کند) از صفات مورد سوال بیشتر باشد. بنابراین می‌تواند به بازیابی داده‌ها و البته در فضای جست-وجوی بزرگتر بپردازد. همانطور که مشهود است شِمای تک جدولی نمی‌تواند این کار را انجام دهد.

DQ9) هدف از این پرس‌وجو، بازیابی مقالاتی است که دو صفت را حتما داشته ولی وجود یک صفت در آنها اختیاری باشد.

به این نوع پرس‌وجوها، پرس‌وجوهای دارای عملگر اختیار گفته می‌شود. روش‌های پیشین که از جدول سه‌گانه مبتنی بر شِمای یکنواخت استفاده می‌کنند، برای پاسخ‌گویی این نوع از پرس‌وجو مجبور به الحاق چپ هستند[Neu10] . اما درروش پیشنهادی با توجه به حضور خوشه‌ها، اگر خوشه-هایی بازیابی شوند که تنها دارای صفات اجباری باشند، جواب پرس‌وجو کامل بازگردانده می‌شود. چرا که همه‌ی صفات موجودیت به همراه او ذخیره شده و اگر آن موجودیت‌ها را بازیابی کنیم، به صفات درخواستی در عملگر اختیار نیز دسترسی داریم. بنابراین بدون توجه به بخش اختیار، موجودیت‌ها بازیابی و سپس صفات درخواست شده درون عملگر اختیار برای هر موجودیت بازیابی می‌شود. این صفات بازگشتی ممکن است برای برخی از موجودیت‌ها *NULL* باشد و برای بعضی دارای مقدار باشد.

اجرای این پرس‌وجو روی شِمای تک جدولی نیز با حذف شدن هزینه الحاق چپ است. این روش نیز تنها دو صفت اجباری را بازیابی می‌کند. همانطور که مشخص است اجرای این دو پرس‌وجو

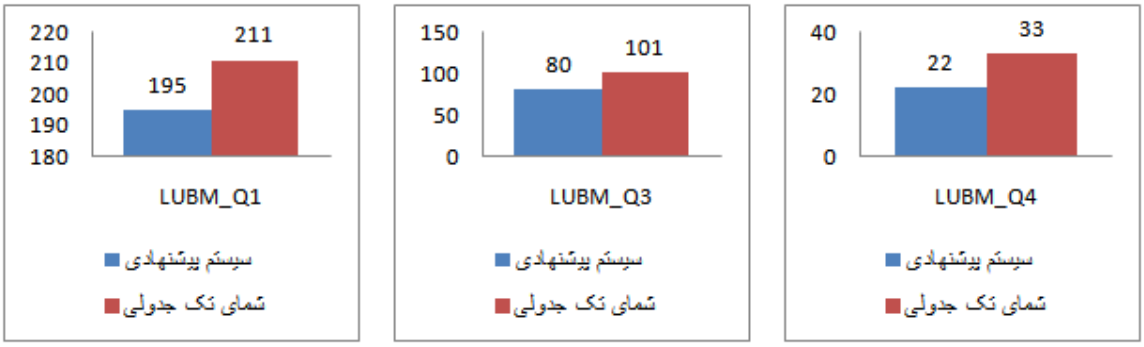
نیز در هردو شِما یکسان است. پس این به این معناست که روش شِمای تک جدولی برای بازیابی داده‌ها تا دو صفت ذکر شده در پرس‌وجو به قدرت روش پیشنهادی عمل می‌کند. علت بالا رفتن هزینه پرس‌وجوی هر دو سیستم نیز نرخ داده‌های بازگشتی است.

DQ10) هدف از این پرس‌وجو یافتن مقالات استناد شده توسط یک موجودیت مقاله‌است.

در این پرس‌وجو یک صفت چند مقداری" استناد" مورد درخواست است. هر دو شِما از امکان برچسب زمانی برای نگهداری صفت‌های(ستون‌ها) چند مقداری استفاده می‌کنند. با توجه به اینکه اولا تعداد صفات درخواستی تنها یک صفت است و ثانیا نرخ داده‌های بازگشتی نیز کم است هر دو شِما موفق عمل می‌کنند.

نتایج پرس‌وجوهای اجرایی روی مجموعه‌داده‌ی LUBM

جداول در این مجموعه‌داده چهار جدول است. نتایج اجرای پرس‌وجوها در شکل(۶-۴) نشان داده‌شده‌است.



شکل ۴-۶: نتایج پرس‌وجوهای اجرایی روی مجموعه‌داده‌یLUBM

اطلاعات کلی از وضعیت پرس‌وجوها و میزان داده‌های بازگشتی را در جدول(۸-۴) می‌توان دید.

---

[1]- Cite

جدول ۴-۸: نتایج حاصل از پرس‌وجوهای اعمالی در مجموعه داده LUBM

پرس‌وجوها	LQ1	LQ2	Lq3
تعداد نتایج	۳	۴	۱۰
تعدادجداول	۱	۱	۱

تحلیل نتایج اجرای پرس‌وجوهای مجموعه‌داده LUBM

LQ1) این پرس‌وجو به دنبال موجودیت‌هایی است که فارغ‌التحصیل شده‌اند و درس GraduateCourse11 رابرداشته‌اند.

در این پرس‌وجو قرار است موجودیت‌هایی با دو صفت بازیابی شوند. از پرس‌وجوهای مجموعه‌داده DBLP نتیجه شد که شِمای تک جدولی تا دوصفت در پرس‌وجوی مورد نظر، همزمان با روش-پیشنهادی عمل می‌کند. بنابراین انتظار می‌رود که در این‌جا نیز به همین شکل باشد. اما همانطور که مشهود است، شِمای تک جدولی در زمان بیشتری به نتیجه رسیده‌است. به این معنا، که هر زمان حجم داده بالاتر شده‌است، صافی مبتنی بر پیشوند در روش پیشنهادی از روش شِمای تک جدولی پیشی می‌گیرد.

LQ2) هدف از این پرس‌وجو یافتن موجودیت‌های منتشر یافته‌ای است که نویسنده‌ی آن AssistantProfessor0 باشد.

روند اجرای این پرس‌وجو دقیقا مشابه پرس‌وجوی Q1) است

LQ3) هدف این پرس‌وجو یافتن موجودیت‌هایی با ۵ صفت است که مقدار دو صفت آن بیان

شده‌است.

روند اجرای این دو پرس‌وجو مانند دو پرس‌وجوی قبل است. اما چرا بازیابی این دو پرس‌وجو با ۵ صفت نسبت به دو پرس‌وجوی دیگر بهتر است؟ در HBase امکان ایجاد صافی سلسله مراتبی وجود ندارد. یعنی نمی‌توان در سطح اول روی هر سطر همزمان صافی پیشوندی و صافی ستون را اعمال کرد و سپس در سطح دوم تمام سطرها را بازیابی نمود. در واقع این‌کار فقط برای یک سطر امکان‌پذیر است. بنابراین برای اعمال صافی سلسله مراتبی، صافی پیشوندی در سرورها روی سطرها اعمال می-شود؛ سپس صافی ستون در سیستم مشتری انجام می‌گیرد. در پرس‌وجوهای Q1) و Q2) این حجم داده‌ی بازگشتی بیشتر بوده و زمان صافی آن در مشتری بیشتر شده‌است. ولی در Q3) چون داده‌های دقیق‌تری بازگردانده‌شده‌است، زمان کمتری طول کشیده‌است.

۵-۲-۴- نتیجه کلی حاصل از ارزیابی سیستم پیشنهادی

۱: روش پیشنهادی نسبت به روش تک جدولی دارای هزینه‌ی سربار استخراج ساختار از داده‌ها است که با استفاده از برنامه‌نویسی نگاشت-کاهش این بخش از سیستم بهینه پیاده‌سازی شده‌است. اما با تحمل این هزینه در پی رسیدن به سه هدف زیر هستیم که تا حد قابل قبولی در بخش بازیابی، این اهداف محقق شده‌اند.

۱. ذخیره‌سازی موجودیت‌های یک خوشه در کنار هم و در نتیجه کمک به بازیابی موثر داده‌های مشابه از نظر ساختار. چرا که HBase در خواندن داده‌ها به صورت ترتیبی نسبت به تصادفی سریعتر عمل می‌کند[Fun12].

۲. امکان جست‌وجوی پشوندی روی سطرها با سرعت بالا

۳. کوچکتر کردن فضای جست‌وجو برای بازیابی دقیق یک موجودیت با تعریف جداول جداگانه متناسب هر الگو

۲: هزینه‌ی ذخیره‌سازی داده‌ها در هر دو سیستم قابل قبول بوده و از تفاوت اندک این دو در نرخ ذخیره‌سازی داده‌ها چشم‌پوشی می‌شود.

۳: روش پیشنهادی با کم کردن فضای جست‌وجو توانسته‌است اجرای پرس‌وجوها را به طور میانگین سریعتر نماید. چرا که برای جست‌وجوی هر الگوی مورد پرس‌وجو به جدولی متناسب با آن رجوع می‌کند.

۴: هر دو روش در پرس‌وجوی یک موجودیت، با یک صفت و یا ذکر دقیق خود موجودیت در زمان یکسانی عمل می‌کنند. در واقع هرچقدر گزینش‌پذیری یک پرس‌وجو بالاتر باشد در نتیجه هر دو روش یکسان عمل می‌کنند.

۵: هنگام پرس‌وجوی یک موجودیت با صفات بیشتر از یکی و با حجم داده‌های بالا، روش پیشنهادی به دلیل استفاده از صافی‌پیشوندی و قرار دادن داده‌ها هم ساختار در کنار هم، از روش مبتنی بر تک جدول پیشی می‌گیرد.

۶: روش پیشنهادی قابلیت پاسخگویی به پرس‌وجوهای با صفات متغیر با شرط وجود حداقل یک صفت ثابت را دارد در صورتی که درشمای تک جدولی این امکان وجود ندارد.

۷: صافی‌های سلسله مراتبی روی جواب پرس‌وجوهای هر دو روش تاثیر زیادی دارند. درسطح اول که داده‌ها در سمت سرورها صاف می‌شوند، این تاثیر مثبت و در سطح دوم که داده‌ها در سمت مشتری صاف می‌شوند این تاثیر منفی خواهد بود.

موارد دیگر کاربرد الگوریتم خوشه‌بندی

همانطور که در بخش (۳-۲-۳) گفته شد این الگوریتم می‌تواند پرطرفدارترین الگویِ توصیفی را مطابق با الگوی مورد پرس‌وجو کاربر پیشنهاد کند. سپس به دسته‌بندی صفات آن الگو بر اساس میزان رخداد آن صفات در الگوهای دیگر بپردازد.

بر این اساس الگوی P با دو عضو، روی مجموعه‌داده‌ی DBLP اعمال می‌شود.

P={#editor,#book_title}

الگوی پیشنهادی سیستم، که شامل الگوی بالا نیز است، الگوی B است که ۴۵۷۵ موجودیتِ کتاب، با این الگو توصیف شده‌اند.

B={#last_modeified_month, #relation, #year, #type, #lable, #volume, #publisher, #in_series, #isbn}

همچنین سه صفت خاص از الگوی B به ترتیب #in_series ، #isbn و #publisher و سه صفت عام پیشنهادی از الگوی B، #type ، #label و #year است. بنابراین این دیدگاه به فرد منتقل می‌گردد که در جست‌وجوی دقیق موجودیتِ کتاب، اولا از چه صفاتی کمک بگیرد و دوما جهت تعریف یک موجودیتِ کتاب، رایج‌ترین نوع تعریف را در دسترس دارد.

۳-۴- خلاصه‌ی فصل

در این فصل ابتدا به نحوه‌ی پیاده‌سازی سیستم پیشنهادی پرداختیم. سپس با معرفی محیط ارزیابی، مجموعه‌داده‌های مورد استفاده را معرفی نمودیم. همچنین با ارائه‌ی چندین معیار ارزیابی در هر بخش از سیستم، شِمای پیشنهادی را مورد ارزیابی با آخرین شِمای ارائه شده قرار داده و به نتایج مطلوب و قابل قبولی در بخش بازیابی رسیدیم.

مراجع

1. [Del10] R. Delbru. “Searching Web Data: an Entity Retrieval Model.” Ph. D. thesis ,National University of Ireland, Ireland, 2010
2. [Mel01] S.Melink, S, Raghavan, B. Yang, H. Garcia-Molina. ” Building a distributed full-text index for the web. ” ACM Transactions on internet Technology, vol. 19, pp. 217-241, jul. 2001
3. [Nar09] A. Narang, V. Agarwal, M. Kedia, V.K. Garg. "Highly scalable algorithm for distributed real-time text indexing," in Proc. of HiPC IEEE , 2009, pp.332-341
4. [Sem] http://www.w3.org/2001/sw/
5. [Fun12] Fundatureanu S.,2012: Scalable RDF Store Based on HBase, MS. D.thesis, Vrije Universiteit, Amsterdam, Netherlands
6. [LOD] Linking open data. http://www.w3.org/wiki/SweoIG/TaskForces /CommunityProjects/LinkingOpenData
7. [Owe08] Owens A., Seab orne A., Gibb ons N., Schraefel, M., 2008: Clustered TDB: A clustered triple store for Jena, Univ. Southampton, Tec. Rep
8. [Neu10] Neumann T., Weikum G., 2010: The rdf-3x engine for scalable management of rdf data, The International Journal on Very Large Data Bases, Vol. 19, Pages 91-113, Feb
9. [RDB] http://readwrite.com/2009/02/12/is-the-relational-database-doomed#awesm=~osqZ3qpcwLCwCq
10. [Med11] Media O., 2011: HBase: The Definitive Guide,
11. [NOSQL] NoSql DataBase: www.nosql-database.org
12. [cha06] Chang F., Dean J., Ghemawat S., Wilson C. Hsieh, Wallach D., Burrows M., Chandra T., Fikes A., Gruber R., 2006: Bigtable: A distributed storage system for structured data, IN PROCEEDINGS OF THE 7TH CONFERENCE ON USENIX SYMPOSIUM ON

OPERATING SYSTEMS DESIGN AND IMPLEMENTATION - VOLUME 7, pages 205-218

13. [Bug12] Bugiotti F., Goasdoué F., Kaoudi Z., 2012: RDF Data Management in the Amazon Cloud, Workshop on Data analyze in Cloud

14. [Gall11] Gallego M., Fernández J., Martínez-Prieto M., Fuente p. 2011: An empirical study of real-world SPARQL queries, In 1st International Workshop on Usage Analysis and the Web of Data (USEWOD2011) at the 20th International World Wide Web Conference (WWW 2011),

15. [Wil06] K. Wilkinson, "Jena property table implementation", International workshop on Scalable Semantic Web Knowledge Base Systems (SSWS) at the International Semantic Web Conference(ISWC), 2006.

16. [Lev09] Levandoski J., Mokbel M., 2009: RDF Data-Centric Storage, In Proceedings of the IEEE International Conference on Web Services, Pages 911-918

17. [Aba07] D. J. Abadi, A. Marcus, S. Madden, and K. J. Hollenbach, "Scalable semantic web data management using vertical partitioning," in proc .*Very Large Data Bases* ,٢٠٠٧ , pp. 1-12.
18. [Ber01] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web," Scientific American, May 2001, pp. 35-34.

19. [Hbase] http://hbase.apache.org/

20. [Had][Hadoop] http://hadoop.apache.org/

21. [TAK][JADID][Empri] P. Cudré-Mauroux, I. Enchev, S. Fundatureanu, P. Groth, A. Haque, A. Harth, F. Keppmann, D. Miranker, J. Sequeda, and M. Wylot. "NoSQL Databases for RDF: An Empirical Evaluation." Proceedings of the 12th International Semantic Web Conference (ISWC). LNCS, vol. 8219, pp. 310-325. Springer, 2013.

22. [Mah12] MahmoudiNasab H., Sakr S.,2012: AdaptRDF: adaptive storage management for RDF databases, , International Journal of Web Information Systems Vol. 8 No. 2,Pages 234-250

23. [Wil03] K. Wilkinson, C. Sayers, H. A. Kuno, and D. Reynolds. "Efficient RDF Storage and Retrieval in Jena2," in Proc. *Semantic Web Data Bases*, 2003, pp. 131-150.

24. [Agg10] C. Aggarwal, H. Wang. “Graph Indexing,” in Managing and Mining Graph Data, 1nd ed., vol. 40, Ed. New York: Springer, 2010, pp. 161-178.

25. [Yan04] X. Yan, P. Yu, and J. Han. “Graph indexing: A frequent structurebased approach,” in Proc *.of the the ACM SIGMOD international conference on Management of data* ,٢٠٠٤ , pp. 335-346 .

26. [Mch97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Wid. “Lore: A Database Management System for Semi-structured Data.” ACM SIGMOD Record,vol. 26, pp. 54 – 66, 1997.

27. [Tha12][Tre12] Tran T., Ladwig G., Rudolph S., 2010: RDF Data Data Partitioning and Query processing Using Structure Indexes, IEEE Trans. Knowledge and Data Engineering, to be published

28. [Wei08] C. Weiss, P. Karras, and A .Bernstein .“Hexastore: sextuple indexing for semantic web data management.” *The International Journal on Very Large Data Bases*, vol. 1, pp. 1008 –1019, 2008.

29. [Kha12] V. Khadilkar, M. Kantarcioglu, B. Thuraisingham, “Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store,” Univ. Texas, Thech. Rep, 2012

30. [Luc12] Lucene.apache.org

31. [Har05] A. Harth and S. Decker, “Optimized index structures for querying rdf from the web " ,in Proc *.of the Third Latin American Web Congress* ,٢٠٠٥ ,pp. 71-81 .

32. [SDB] Apache jena: www.jena.apache.org

33. [Ses] Sesame: www.openrdf.org

34. [Har07] Harth A., Umbrich J., Hogan A., Decker S., 2007: YARS2: A Federated repository for Searching and Querying Graph Structured Data, In Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference, Pages 211-224

35. [H2RDF][Pap12] Papailiou N., Konstantinou I., Tsoumakos D., Koziris N., 2012: H 2RDF: Adaptive Query Processing on RDF Data in the Cloud, *In proceedings of the 21st International World Wide Web Conference*

(WWW2012 Demo Track), Pages16-20, Apr

36. [Hog12][Hog11] Hogan A., Harth A., Umbrich u., Kinsella S ,.Polleres A., Decker S., 2011: Searching and Browsing Linked Data with SWSE: the Semantic Web Search Engine, The International Journal on Web Semantics: Science ,Services and Agents on the World Wide Web, Elsevier Science, Vol. 9, Pages ,٤٠١-٣٦٥ Des

37. [Siv12] Siva M., Poobalan A., 2012: Semantic Web Standard in Cloud Computing, International Journal of Soft Computing and Engineering (IJSCE), Vol. 1, Jan

38. [Mat05] A. Matono, T. Amagasa, M. Yos hikawa, and S. Uemura. "path-based relational RDF database," in Proc *.of the 16th Australasian database conference*, ٢٠٠٥ ,pp .١٠٣-٩٥ .

39. [Udr07] O. Udrea, A. Pugliese, and V. Subrahmanian,"Grin: a graph based rdf index", AAI Conference of Artifical Inteligent , vol. 22, no. 2, 2007.

40. [Min08] E. Minack,L. Sauermann,G. Grimnes,C. Fluit, J. Broekstra, "The Sesame uceneSail: RDF Queries with Full-text Search," Tech. Rep. Nepomuc, 2008.

41. [Bro03] J. Broekstra, A. Kampman, F. Harmelen .*Sesame :An architecture for storing and querying RDF data and schema information* .Spinning the Semantic Web, 2003 .

42. [Neu11] T. Neumann, A. Gubichev. "Path Query Processing on Very Large RDF Graphs," in *Proc. WebDB*, 2011, pp. 1-6

43. [Sun10] Sun j., 2010: Scalable rdf store based on hbase and mapreduce, 3rd International Conference In Advanced Computer Theory and Engineering (ICACTE)

44. [Wan10] X. Wang, S. Wang, P. Du, Z .Feng ."Storing and Indexing RDF Data in a Column-Oriented DBMS," in Proc *.of Database Technology and Applications (DBTA* ,٢٠١٠ ,*(*pp. 1-4 .

45. [Ver12] Vertica: www.vertica.com

a. [Clus] *Hierarchical clustering*, http://en.wikipedia.org/wiki/Hierarchical_clusteringLP .

46. [LOG] Kramer, Kasjen, Renata Dividino, and Gerd Gröner. "SPACE: SPARQL Index for Efficient Autocompletion."

47. [Lubm] http://swat.cse.lehigh.edu/projects/lubm/

48. [Dblp] *The DBLP Computer Science Bibliography* ,http://dblp.uni-trier.de· /