ارائهٔ روشی پویا جهت پاسخ به پرس‌وجوهای پیوسته تجمّعی اقتضایی

مهدی مسافری[1] و علی‌اصغر صفائی[2]*

[1]شرکت جویا افزار ماندگار پرسیا، تهران، ایران

[2]گروه انفورماتیک پزشکی، دانشکده علوم پزشکی، دانشگاه تربیت مدرس، تهران، ایران

چکیده

جریان‌های داده دنباله‌های نامتناهی، سریع، متغیر با زمان و با نرخ ورود انفجاری از عناصر داده هستند که به‌طورمعمول نیاز دارند به‌صورت برخط و به‌طورتقریبی بی‌درنگ پردازش شوند. بر این اساس، الگوریتم‌های پردازش جریان‌های داده و اجرای پرس‌وجوها روی جریان داده‌ها بیش‌تر تک‌گذره هستند. اجرای این الگوریتم‌های تک‌گذره با محدودیت‌ها و چالش‌هایی از قبیل محدودیت در حافظه، زمان‌بندی، و دقت پاسخ‌ها مواجه است. این چالش‌ها به‌ویژه در شرایطی که پرس‌وجوی مورد نظر از قبل تعیین و مشخص نشده باشد و به‌صورت اقتضایی، پس از ارسال جریان داده ارائه شود، به‌مراتب جدی‌تر و حل آن‌ها دشوارتر خواهد بود. در این مقاله، برای پردازش پرس‌وجوهای تجمعی که به‌طور پیوسته روی جریان‌های داده اجرا خواهند شد و البته به‌طور اقتضایی ارائه می‌شوند، راه حلی مبتنی بر ساختار درختواره و نگهداشت نتایج تجمعی معرفی شده است. نکته مهم در این روش، برقراری برخط بودن در تمام مراحل ساخت، نگهداری و بهره‌برداری از درخت است. برای تأمین برخط بودن فرایند پاسخ به پرس‌وجو، کافی است تمامی پاسخ‌های محتمل را نگهداری کنیم؛ اما برای حفظ برخط‌بودن فرایند ساخت و نگهداری درخت، با توجه به ویژگی‌های ذاتی جریان داده ناچاریم برخی پاسخ‌ها را نگهداری کنیم. بدین ترتیب، هدف و مسئله اساسی آن است که دست‌کم پاسخ‌های انتخابی برای ذخیره در قالب درختواره را به مجموعه پاسخ‌های مورد نیاز برای پرس‌وجوهای اقتضایی رسیده نزدیک‌تر کنیم. ساختار درخت تجمعی پیشوندی پیشنهادی که به‌صورت پویا ایجاد، نگهداری، مدیریت و در پردازش پرس‌وجوها استفاده می‌شود، تشریح و صحت عملکرد آن به‌صورت عملی مورد ارزیابی قرار گرفته که نتایج حاکی از کارآمدبودن آن برای به‌کارگیری در پردازش برخط پرس‌وجوهای پیوسته تجمعی اقتضایی روی جریان‌های داده است.

واژگان کلیدی: پرس‌وجوی پیوسته تجمعی اقتضایی، جریان داده، درخت پیشوندی پویا، سلول تجمعی.

Providing a Dynamic Technique for Answering Ad-hoc Continuous Aggregate Queries

Mehdi Mosaferi[1] & Ali Asghar Safaei[2]*

[1]JAM Persia Company, Tehran, Iran

[2]Department of Biomedical Informatics, Faculty of Medical Sciences, the University of Tarbiat Modares, Tehran, Iran

Abstract

Data Streams are infinite, fast, time-stamp data elements which are received explosively. Generally, these elements need to be processed in an online, real-time way. So, algorithms to process data streams and answer queries on these streams are mostly one-pass. The execution of such algorithms has some challenges such as memory limitation, scheduling, and accuracy of answers. They will be more important and serious, chiefly if the queries are not predefined but Ad-hoc, and also should be executed after data stream tuples are gone.

Countinous aggregate queries are types of queries with some special characteristics making it possible to perform more specific, efficient qeury processing techniques, specifiaclly beneficient for ad-hoc ones.

* نویسندهٔ عهده‌دار مکاتبات

* Corresponding author

In this paper, a dynamic efficient techinque is proposed for answering the ad-hoc continiues aggregate queries over data streams. The main idea of the proposed technique is to generate and handle an efficiet tree data structure as the synopse, in the form of Dynamic Prefix Aggregate Tree.

In general, the two following approaches can be used to calculate any function such as $f(x, y, z)$; either implementation of an algorithm for the calculation of function f, or storing the answers of function f for all possible states. When the algorithm runtime is high, the second method strengthened by proper selection of indices can return a proper answer in a very short time (even $O(1)$).

But the major problem of the second method is the total number of possible answers which can be very high and also can be out of the possible storage capacity and processing potential within a certain acceptable time period. For example, suppose that the cardinality of each of the parameters of x, y, z is 10. In this case, the total number of possible states will be $11^3 = 1331$. As it is evident, the total number of states increases with the number of parameters and their cardinalities.When the total number of states is so great that generating answers with respect to consumed time and space is impossible, a more convenient, practical method should be employed. This more practical approach can be the storing of some of the answers (selectively) with respect to the following conditions:

- **Obtaining un-stored answers from the set of stored answers.**
- **Higher probability of utilizing stored answers (i.e. higher probability of submitting requests from stored set).**
- **Eliminating (not storing) null answers.**

The same idea can be implemented for online and almost real time processing of queries, so that by receiving each tuple, all possible answers get obtained and stored. By doing so, in the time of need (when answering to an ad-hoc query) stored answers will be used instead of calculating each answer.

Accordingly, some answers are stored in a tree structure to be used at the right time. In this paper, in order to answer ad-hoc continuous aggregate queries over data streams, a method is proposed that uses a tree structure for storing the aggregate results. The important point in this method is that all steps of the construction, maintenance and using of the tree must be online. For these purposes, it is enough to keep all possible answers. But to apply an online construction and maintenance of tree, we must keep some answers, according to the inherent features of data streams. In this way, the main goal is to choose the answers possessing the most overlap with responses answers of received ad-hoc queries. The proposed method, creates the tree structure and maintains it dynamically to answer ad-hoc aggregate continuous queries over data streams.

For this purpose, queries at instant τ are modeled as in form of $(\tau, d_1, \dots, d_n)$, where $t - \omega + 1 \leq \tau \leq t$ or $\tau = *$ (when $\tau = *$, the aggregate over the whole sliding window is returned) and ω is the size of sliding window and $d_i \in D_i \cup \{*\}$ (when $d_i = *$, the aggregate over the whole D_i is returned).

In order to increase the overlapping, a statistical task is performed on a dimensions of the received queries. In this way, dimensions are determined with the highest, lowest request. When $d_i = *$, means that there is no request for this dimension. Therefore, we select and store the answers related to the dimension with highest request, and ignore those with the lowest. Obviously, these answers should be obtained and presented using stored answers.

As the request for dimensions may change, the tree structure must be dynamically constructed and maintenance that will be presented this dynamic structure in this paper. Experimental evaluattion of the proposed method shows that, using the proposed Dynamic Aggregate Tree for ansering countinous Ad-hoc aggregate queies is more cost-effective, in terms of response time and memory usage.

Keywords: Data Stream, Continuous ad-hoc aggregate queries, Dynamic Prefix Tree, Aggregate cell.

۱- مقدمه

امروزه کاربردهای متعددی هستند که به‌طورذاتی داده‌ها را در قالب جریان داده‌ها ایجاد یا پردازش می‌کنند. شبکه‌های حس‌گر، تحلیل نوسانات مالی نظیر بورس، پایش‌ها نظیر پایش ترافیک شبکه‌ها، پایش مراقبت سلامت، ثبت رکوردهای تماس‌های تلفنی یا جریان کلیک‌ها در وب، تنها نمونه‌هایی از کاربردهای فراوانی از این دست هستند؛ به‌عبارت دیگر، آنچه امروزه تحت عنوان داده‌های عظیم[1] یا کلان‌داده‌ها به‌عنوان چالش اصلی و مهم در حوزه مدیریت داده‌ها شناخته می‌شود، یکی از ابعادش سرعت بالای ورود و خروج (پردازش) داده‌ها است که مصداق آن همین جریان داده‌ها است [1].

مسئله داده‌های عظیم، مسئله استخراج ارزش مستتر در مجموعه دادهایی است که ابعاد پیچیدگی آنها نسبت به گذشته افزایش یافته و دست‌کم شامل سه بُعد حجم[2]، تنوع[3]،

[1] Big data

[2] Volume

[3] Variety

و سرعت[1] (3Vs) است. منظور از این سه بعد پیچیدگی به‌ترتیب: حجم بالای تولید داده در حد روزانه چندین پتابایت، تنوع در منابع تولیدکننده و نوع داده‌ها از جمله از نظر میزان ساخت‌یافتگی، و سرعت بالا هم در ورود داده‌ها و هم در پردازش داده‌ها. جریان داده‌ها شناخته‌شده‌ترین مصداق از سرعت بالای ورود داده‌ها معرفی می‌شود [2].

جریان داده، دنباله‌ای پیوسته، نامتناهی، سریع، متغیر با زمان و شاید غیرقابل پیش‌بینی از عنصرهای داده است که همواره به انتهای آن افزوده می‌شود [3]. هر عنصر داده می‌تواند یک مقدار ساده (مثل یک عدد صحیح که حاصل مقدار خوانده‌شده توسط یک حس‌گر است) باشد یا یک تاپل[2] (چندتایی) حاوی داده از یک رابطه (مانند ثبت مکالمات تلفنی). برای حفظ ترتیب عنصرها در این دنباله، به‌طورمعمول در کنار هر عنصر یک برچسب زمانی قرار می‌گیرد که اغلب توسط منبع تولیدکننده داده تعیین می‌شود. فرمت عمومی داده‌های روی جریان داده به‌صورت زوج مرتب (s,t) است که در آن s تاپل داده و t برچسب زمانی این عنصر داده تعریف می‌شود. سامانه‌های مدیریت پایگاه‌های داده[3] (DBMS) سنتی و مرسوم قابلیت پردازش جریان‌های داده با این ویژگی‌ها را ندارند و نیاز به سامانهٔ خاصی برای مدیریت جریان داده داریم که DSMS[4] نام دارد.

سامانه‌های مدیریت پایگاه داده برای کاربردهای سنتی که اغلب تجاری بوده و در آن‌ها داده‌ها در یک پایگاه ذخیره شده و پرس‌وجو به‌صورت لحظه‌ای روی مقادیر آن لحظه پایگاه داده اعمال می‌شود، مفید خواهند بود؛ اما امروزه کاربردهای فراوانی ارائه شده‌اند که دیگر داده‌ها به‌صورت مانا در یک پایگاه ذخیره نمی‌شوند، بلکه در قالب دنباله‌های نامتناهی از عنصرهای داده هستند که به‌صورت گذرا روی جریان‌ها وارد می‌شوند و ازآنجاکه حجم آن‌ها نامتناهی است، امکان ذخیره‌سازی همه آن‌ها در حافظه‌های جانبی وجود ندارد؛ لذا باید بتوان آن‌ها را به‌صورت برخط و بی‌درنگ پردازش کرد. چنین پردازشی با این قابلیت‌های متمایز و جدید روی جریان‌های داده، استفاده از سامانه‌های مدیریت پایگاه داده مرسوم امکان‌پذیر نیست و نیاز به سامانه‌های جدیدی به نام سامانهٔ مدیریت جریان داده داریم.

پرس‌وجوهایی که در سامانه‌های مدیریت جریان داده اجرا می‌شوند، متفاوت از پرس‌وجوهای سنتی سامانه‌های مدیریت پایگاه داده بوده و می‌توانند انواع زیر را شامل شوند:

- پرس‌وجوهای **پیوسته**[5] در مقابل پرس‌وجوهای یک‌باره[6]
- **پرس‌وجوهای یک‌باره** فقط یک‌بار روی جریان داده اجراشده پاسخ گرفته و خاتمه می‌یابند، اما پرس وجوهای پیوسته از زمان ثبت در سامانه به‌طور مستمر و پیوسته روی جریان داده ورودی اجرا می‌شوند و پاسخ برمی‌گردانند (مادامی‌که جریان داده وجود داشته باشد و دستور خاتمه پرس‌وجو داده نشود).
- **پرس‌وجوهای اقتضایی**[7] در مقابل پرس‌وجوهای **از** پیش تعریف‌شده[8]
- **پرس‌وجوهای ازپیش‌تعریف‌شده،** قبل از شروع‌شدن جریان داده در سامانه ثبت شده‌اند؛ اما پرس‌وجوهای اقتضایی پس‌ازآنکه جریان داده شروع به ارسال‌شده و بخشی از آن گذشته است، به سامانه ارائه و در آن ثبت می‌شوند؛ لذا نگهداری آنچه این پرس‌وجو باید از داده‌های پیشین ازدست‌رفته بداند (با توجه به اینکه نمی‌دانیم چه پرس‌وجوهایی ممکن است در آینده مطرح شوند) بسیار چالش‌برانگیز است. از طرفی، یک پرس‌وجو فارغ از اینکه پیوسته باشد یا یک‌باره، می‌تواند اقتضایی باشد یا ازپیش تعریف‌شده، و البته در معنای[9] خود ممکن است از عملگرهای گوناگون مورد نیاز استفاده کند.

در این مقاله، اجرای پرس‌وجوهای پیوستهٔ اقتضایی که از نوع تجمعی بوده و در آن‌ها عملگرهای تجمعی نظیر max، min، sum، avg و count به‌کار رفته است، مورد نظر هستند. این پرس‌وجوها به‌واسطهٔ اقتضایی‌بودن، بسیار چالش‌برانگیزند و البته به‌دلیل تجمعی و پیوسته‌بودنشان خلاصه‌هایی از نتایج تجمعی گذشته و تاکنون را برای پاسخ

---

[1] Velocity

[2] Tuple

[3] Data Base Management System

[4] Data Stream Management System

[5] Continuous

[6] One-time

[7] Ad-hoc

[8] predefined

[9] semantic

در آینده می‌توان استفاده کرد. در این مقاله، ساختاری درختواره برای این منظور ارائه شده است که به‌صورت پویا ایجاد، نگهداری و در زمان پاسخ به پرس‌وجوها استفاده می‌شود.

از مهم‌ترین چالش‌هایی که در پردازش این پرس وجوها روی جریان داده‌ها وجود دارد، به موارد زیر می‌توان اشاره کرد [2]:

- پردازش باید برخط و به‌طورتقریبی بی‌درنگ باشد.
- الگوریتم‌های مورد استفاده برای پردازش باید تک‌گذره بوده و هر داده را حداکثر یک‌بار می‌توان بررسی کرد و پس‌ازآن، داده از بین می‌رود.
- محدودیت در اندازه حافظه وجود دارد.
- پاسخ‌ها به‌طورمعمول تقریبی هستند.

با وجود این چالش‌ها و محدودیت‌ها باید راه‌حلی برای پردازش پرس‌وجوهای پیوسته تجمعی اقتضایی ارائه کرد که بتواند برخط و تک‌گذره خلاصه‌هایی تجمعی از داده‌ها را حفظ و نگهداری کند تا درصورت ثبت پرس‌وجوهایی اقتضایی بتواند به‌صورت **برخط** پاسخ قابل قبولی (از نظر **دقت** و **کامل‌بودن**) ارائه کند.

به‌طورکلی، برای محاسبه هر تابعی نظیر $f(x,y,z)$ دو راهکار زیر متصور است:

- پیاده‌سازی الگوریتم محاسبه تابع f
- ذخیره‌کردن پاسخ تابع f به ازای تمام حالات ممکن

در شرایطی که زمان اجرای الگوریتم بالا باشد، روش دوم با یک شاخص‌گذاری مناسب در زمان بسیار کوتاه (حتی $O(1)$) پاسخ مناسب را می‌تواند برگرداند. البته مشکل مهم روش دوم این است که تعداد کل حالات پاسخ ممکن، بسیار زیاد بوده و خارج از توان ذخیره و پردازش در زمان قابل قبول باشد. به‌عنوان‌مثال فرض کنید کاردینالیتی[1] هر یک از پارامترهای x,y,z مقدار ۱۰ باشد. در این صورت تعداد کل حالات ممکن $11^3 = 1331$ خواهد بود. همان‌طور که مشخص است، تعداد کل حالات با افزایش تعداد پارامترها و کاردینالیتی آن‌ها افزایش می‌یابد. اگر تعداد کل حالات به‌قدری زیاد باشد که از نظر زمان و فضای مصرفی امکان تولید پاسخ‌ها وجود نداشته باشد، باید روش مناسب و عملی تری را استفاده کرد.

روش عملی‌تر نگهداری برخی از پاسخ‌ها (به‌صورت گزینشی) می‌تواند باشد؛ به‌گونه‌ای که شرایط زیر برقرار باشند:

۱. بتوان پاسخ‌های ذخیره‌نشده را از مجموعه پاسخ‌های ذخیره‌شده به‌دست آورد.
۲. احتمال استفاده از پاسخ‌های ذخیره‌شده بیش‌تر باشد. (یعنی احتمال درخواست‌ها از مجموعه ذخیره‌شده، بیشتر باشد)
۳. پاسخ‌های تهی ذخیره نمی‌شوند.

در [4] نیز برای بهبود سرعت پاسخگویی، با استفاده از پرس‌وجو (درخواست)های رسیده قبلی، و با کمک از الگوریتم Index-BittableFI دیدهای پرتکرار را یافته و آنها را ذخیره می‌کند. درواقع با انتخاب گزینش از ذخیره تمامی دیدها صرف‌نظر کرده است.

برای پردازش برخط و به‌طورتقریبی بی‌درنگ پرس‌وجوها روی جریان داده‌ها نیز با ایده‌ای مشابه با رسیدن هر تاپل، تمامی پاسخ‌های ممکن را می‌توان به‌دست آورده و نگهداری کرد. بدین ترتیب در زمان لزوم (پاسخ به یک پرس‌وجوی اقتضایی)، به‌جای محاسبه پاسخ‌ها، از مقدار ذخیره‌شدهٔ آن‌ها استفاده می‌شود. مفهوم درخت تجمعی پیشوندی[2] [5] نیز بر اساس همین روش پیشنهاد شده است. است.

در روش پیشنهادی در این مقاله، با اصلاح ساختار درخت به‌صورت پویا، تعداد پاسخ‌های مؤثر ذخیره‌شده افزایش خواهد یافت. بدین ترتیب، ضمن ذخیرهٔ بخشی از پاسخ‌ها به‌صورت گزینشی (به‌جای همه پاسخ‌ها)، موازنهٔ قابل قبولی بین دقت پاسخ‌های ارائه‌شده از یک‌سو، و کارآمدی سامانه (شامل پاسخ‌گویی برخط و بهره‌وری منابع و حافظه مصرفی) از سوی دیگر ایجاد می‌شود.

ساختار ادامه مقاله بدین‌صورت است که در بخش ۲، پس‌زمینه موجود درخصوص نحوهٔ ساخت و نگهداری درخت تجمعی پیشوندی، و چگونگی پاسخ به پرس‌وجوها بیان می‌شود. در بخش ۳ تشریح روش پیشنهادی درخت تجمعی پیشوندی پویا و چگونگی ایجاد، نگهداشت و به‌کارگیری درخت برای پاسخ به پرس‌وجوها ارائه شده است. ارزیابی عملی روش پیشنهادی و پیچیدگی‌های آن در مقایسه با سایر روش‌های مشابه در بخش ۴ انجام شده است. بخش ۵ به معرفی برخی از کارهای مرتبط پرداخته و درنهایت نتیجه گیری و پیشنهاد برای برخی از کارهای آتی در بخش ۶ آمده است.

[1] Cardinality = تعداد اعضای مجموعه دامنه.

[2] Prefix Aggregate Tree (PAT)

۲- پیش‌زمینه: درخت تجمعی پیشوندی

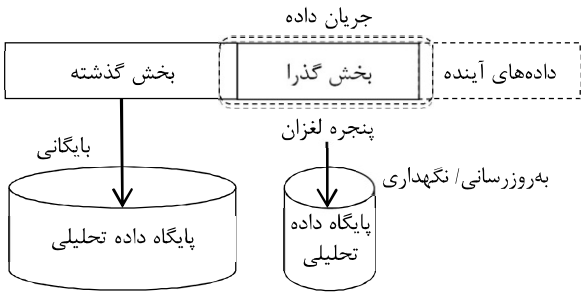
پایگاه داده تحلیلی[1] و پردازش تحلیلی برخط[2] برای بسیاری از کاربردها و وظایف تحلیل داده‌ای، جزء امکانات ضروری محسوب می‌شوند. برای یک جدول پایهٔ داده‌شده، یک پایگاه داده تحلیلی، یک مجموعه بزرگی از تجمعات عینی‌سازی شده[3] است. با استفاده از یک شاخص‌گذاری[4] مناسب روی پایگاه داده تحلیلی، می‌توان پرس‌وجوهای تجمعی متنوعی را به‌صورت برخط پاسخ داد. از آنجایی که مجموعهٔ کامل تجمعات روی ابعاد جدول پایه[5] (جداول اصلی برای ذخیره داده‌های پایگاه‌داده)، به‌طورمعمول بسیار بزرگ است، در [5] ساختار داده‌ای درخت تجمعی پیشوندی پیشنهاد شده است.

۲-۱- چارچوب نگهداری اطلاعات

در این چارچوب، جریان داده به‌صورت یک جدول پایهٔ بی‌انتهای $S(T, D_1, \cdots, D_n, M)$ مدل می‌شود که در آن T مهرهای زمانی، $D_1, \cdots, D_n$، n بعد در دامنه‌های گسسته و M، مقدار[6] تعریف شده است. برای سادگی کار، مهرهای زمانی را اعدادی صحیح با شروع از یک، در نظر می‌گیریم. درست است که پایگاه‌های داده تحلیلی می‌توانند به پرس‌وجوهای تجمعی متنوعی به‌صورت کارآمدی پاسخ دهند، ولی این پایگاه‌های داده تحلیلی نیاز است که به‌صورت مداوم و دوره‌ای و اغلب برون‌خطی، به‌روزرسانی شوند. به عبارتی پاسخ به پرس‌وجوهای تجمعی‌ای که داده‌های آن‌ها به‌تازگی رسیده و هنوز در پایگاه اعمال نشده‌اند، سخت است. برای چیره‌شدن بر این مشکل، یک جریان داده را با توجه به شکل (۱) به دو بخش تقسیم می‌کنیم: بخش گذشته[7] و بخش گذرا[8].

همان‌طور که مشخص است، داده‌های بخش گذشته، قبل از آخرین به‌روزرسانی پایگاه داده تحلیلی مرکزی رسیده‌اند و بنابراین بایگانی[9] شده‌اند، درعوض، داده‌های بخش گذرا، هنوز در پایگاه، بایگانی نشده و باید در یک پایگاه

[1] Data warehousing
[2] Online analytic processing (OLAP)
[3] materialized
[4] index
[5] Base table
[6] Measure
[7] Historical segment
[8] Transient segment
[9] Archive



(شکل-۱): چارچوب پایگاه داده تحلیلی برای جریان‌های داده [5]
(Figure-1): The framework of warehousing data streams

داده تحلیلی برخط به‌روزرسانی و نگهداری شود.

مهر زمانی تاپل‌های بخش برخط چارچوب، به یک پنجره لغزان به‌اندازهٔ ω تعلق دارند. به‌عبارت دیگر فرض می‌کنیم در لحظهٔ t، $(t \geq \omega)$، پرس‌وجوهای تجمعی پایگاه داده تحلیلی برخط، همواره پیرامون تاپل‌های پنجره لغزان $[t-\omega+1, t]$ واقع شده‌اند (این فرض مشکلی ایجاد نمی‌کند. چون پرس‌وجوهای دیگر روی بخش گذشته بوده و این اجرا توسط یک پایگاه داده تحلیلی قابل انجام است). توابع تجمعی یکی از توابع SUM، MIN، MAX، AVG و COUNT می‌توانند باشند.

۲-۲- ساختار داده‌ای[10]

جریان دادهٔ $S(T, D_1, \cdots, D_n, M)$ را در نظر بگیرید. ساختار داده، به‌صورت یک درخت تعریف می‌شود که در آن هر گرهٔ میانی، مقادیر تجمعی تمام تاپل‌هایی را که نسبت به ریشه، پیشوند یکسانی دارند، نگه می‌دارد. به این گره‌ها، سلول‌های تجمعی پیشوندی[11] گفته می‌شود.

تعریف ۲-۱ (سلول تجمعی پیشوندی): جریان دادهٔ S را در نظر بگیرید. برای هر تاپل، فهرستی از ابعاد به‌ترتیب $D_1, \cdots, D_n$ وجود دارد. S_t نیز شامل تاپل‌های پنجره لغزان جاری (یعنی بین بازه $t-\omega+1$ تا t) از جریان S هست. یک سلول تجمعی $c = (\tau, d_1, \cdots, d_n)$ به شرطی یک سلول تجمعی پیشوندی است که: نخست بین ۱ و n یک kای وجود داشته باشد که تمام ابعاد $d_1, \cdots, d_k$ غیر * و تمام ابعاد بعدازآن؛ همه * باشند. دوم‌این‌که دست‌کم یک تاپل مانند $c' = (d'_1, \cdots, d'_n)$ در S_t وجود داشته باشد که برای همه iهای کوچک‌تر و مساوی k، $d'_i = d_i$ برقرار باشد.

[10] Data structure
[11] prefix aggregate cells

سلول‌های تجمعی پیشوندی را با استفاده از عبارت منظم[1] به‌صورت $\nabla^* * \nabla^*$ تعریف می‌شود. در این تعریف حروف الفبا $\Sigma = \{\nabla, *\}$ بوده که ∇ نماینده تمامی مقادیر غیر $*$ است.

- **تعریف ۲-۲ (درخت تجمعی پیشوندی):** به‌طورکلی برای پنجره لغزان جاری داده‌شده روی یک جریان داده، یک PAT، یک درخت پیشوندی از سلول‌های تجمعی پیشوندی به همراه دو نوع یال زیر تعریف می‌شود:
- **یال‌های کناری**[2]: همه گره‌های یک بعد را که برچسب یکسان دارند به هم وصل کرده و به‌صورت یک مشخصه محلی نگهداری می‌کنیم. (به‌صورت زنجیره‌ای به هم وصل می‌شوند)
- **یال‌های میانوندی:**[3] گره $v = d_1 \cdots d_k$ در نظر بگیرید $(1 \le k \le n-2)$. برای هر مقدار $d_{i,j}$ در بعد D_i (یعنی برای بعد i ام تعداد j مقدار متمایز وجود دارد) در زیردرختی به ریشهٔ v ($k+2 \le i \le n$)، همهٔ گره‌های این زیردرخت که برچسب $d_{i,j}$ دارند، توسط یال‌های کناری به‌صورت زنجیره‌ای به هم وصل شده‌اند. یک یال میانوندی از گره v به نخستین عضو این زیر فهرست (نودهایی که توسط یال کناری به هم وصل شده‌اند) ایجاد، و یک جدول تجمعی را مربوط به این یال ذخیره می‌کنیم. این یال مقدار $c = (d_1, \cdots, d_k, *, \cdots, *, d_{i,j}, *, \cdots, *)$ را نگه می‌دارد. به c یک سلول تجمعی میانوندی گفته می‌شود. با استفاده از عبارت منظم $\nabla^* * + \nabla^* *$ می‌توان یک سلول تجمعی میانوندی را تعریف کرد ($\nabla^* * \nabla^* * \nabla^*$ در یک حالت سلول تجمعی پیشوندی خواهد بود).

۲-۳- ساختن درخت تجمّعی پیشوندی (PAT)

برای ساختن PAT، تاپل‌ها را یکی‌یکی (و یا دسته‌دسته) به‌گونه‌ای به حافظه اصلی منتقل می‌کنیم که هر تاپل فقط یک‌بار خوانده شود. PAT را با یک ریشه مقداردهی اولیه می‌کنیم؛ سپس نخستین تاپل را خوانده و در درخت درج می‌کنیم. برای هر گرهٔ مسیر، یک سطر نیز در جدول تجمعی ثبت می‌شود. یال‌های میانوندی و جداول تجمعی متناظر آن‌ها نیز ایجاد می‌شوند. به‌محض اینکه اطلاعات در درخت ثبت شد، دیگر نیازی به تاپل نیست. به همین ترتیب تاپل بعدی نیز خوانده و در درخت درج می‌شود. برای گره‌هایی که ازقبل درج شده‌اند، تنها مقادیر تجمعی آن‌ها به‌روز می‌شود.

ساختن یک PAT با استفاده از پویش یکی‌یکی تاپل‌ها، دو مؤلفه اصلی دارد: ساختن درخت پیشوندی و دیگری ایجاد و نگهداری یال‌های کناری و میانوندی. تجمع در لحظه t باید در سطر $(t \bmod \omega)$ از جداول تجمعی ذخیره شود. بنابراین هزینه نگهداری و جستجو هم در جدول، مقدار ثابتی است. همچنین لازم است با اضافه‌شدن تاپل‌های لحظه $(t+1)$، به درخت، داده‌های لحظه $(t-\omega+1)$ از درخت حذف شوند. با درج تاپل‌های جدید، یک رکورد در محل $((t+1) \bmod \omega)$ از جدول تجمعی تمامی گره‌های مسیر ذخیره می‌شود. این سطر به‌طور خودکار جایگزین سطر $(t-\omega+1)$ می‌شود. خاطر نشان می‌شود که این حذف به‌عنوان یک اثر جانبی درج انجام شده است. یال‌های میانوندی و کناری، متناسب این گره‌ها و همچنین جداول تجمعی آن‌ها نیز تنظیم می‌شوند. در ادامه برای حذف سایر گره‌ها از متغیر LUT[4] استفاده می‌شود. برای هر برگ یک عدد صحیح به نام LUT (آخرین زمان به‌روزرسانی) نگهداری می‌کنیم. همه گره‌های برگی را که LUT یکسانی دارند به‌صورت یک فهرست پیوندی به هم وصل می‌کنیم. به سراغ فهرست پیوندی که $LUT = t-\omega+1$ دارند، می‌رویم. از این گره‌ها به سمت ریشه حرکت کرده و آن‌قدر ادامه می‌دهیم تا به نخستین گره‌ای برسیم که در جدول تجمعی‌اش، سطری به‌غیراز لحظه $t-\omega+1$ داشته باشد. لازم است تمامی این گره‌ها را در مسیر حرکتمان حذف کنیم. البته با این روش ممکن است سطرهایی از لحظه $(t-\omega+1)$ هنوز باقی بماند که حذف این سطرها به آینده موکول می‌شود.

۲-۴- پاسخ به پرس‌وجوی تجمّعی

این پرس‌وجوها می‌توانند به‌صورت مؤثری با استفاده از یک PAT پاسخ داده شوند، روش کلی کار به‌شرح زیر است:

- اگر سلول تجمعی، پیشوندی یا میانوندی باشد، پاسخ به‌حتم در جداول تجمعی درخت ذخیره شده و به‌راحتی به‌دست می‌آید.

[1] Regular expressions

[2] side link

[3] infix links

[4] Last update time-stamp

- در غیر این صورت سلول تجمعی به چندین سلول تجمعی پیشوندی تبدیل شده و با استفاده از تجمع پاسخ‌های به‌دست آمده، پاسخ نهایی محاسبه می‌شود.

در ساختارهای داده ایستا لازم است یک الگوی پیش‌فرض برای ترتیب ابعاد انتخاب شود که ممکن است با درخواست‌های رسیده مطابقت نداشته باشد.

این ضعف بزرگی برای روش مبتنی بر درخت تجمعی پیشوندی محسوب می‌شود که کارآمدی آن را در شرایط واقعی به چالش می‌کشد.

۳- روش پیشنهادی: درخت تجمّعی پیشوندی پویا

با توجه به طبیعت پویای جریان داده‌ها و نامتناهی‌بودن آن، ساختار داده ایستا به مرورزمان کارامدی خود را (از نظر حافظه مصرفی و زمان انجام عملیات) از دست خواهد داد و حتی ممکن است قادر به ارائه کارکردهای موردنیاز نباشد؛ لذا با توجه به اینکه ایستابودن ساختار داده مورد استفاده به‌لحاظ نظری و عملی با طبیعت جریان داده در تعارض است، ایده اصلی در روش پیشنهادی، به‌کارگیری ساختاری پویا به‌جای ایستا و طراحی تمهیدات و ملزومات مربوطه است.

یک سلول پرس‌وجو مانند (T, A, B, C, D) را در نظر بگیرید که در آن T مهر زمانی بوده و یک عدد در محدوده پنجره و یا * می‌تواند باشد. هر یک از ابعاد D, C, B, A نیز یک مقدار از دامنه خود و یا مقدار * می‌توانند داشته باشند.

به‌عبارت دیگر هر یک از این اجزا مقدار * و یا غیر* می‌توانند داشته باشند. یک سلول پرس‌وجو بر اساس ترکیب چینشی مقادیر * و غیر * در کنار هم، یکی از حالات زیر می‌تواند باشد:

- سلول پرس‌وجوی پیشوندی
- سلول پرس‌وجوی میانوندی
- هیچ‌کدام

برای به‌دست‌آوردن پاسخ پرس‌وجوی حالت‌های نخست و دوم، از جداول تجمعی ذخیره‌شده در درخت استفاده می‌کنیم. در این حالت تنها کافی است، جدول مربوطه را پیدا کنیم؛ ولی برای حالت سوم لازم است، سلول را به چندین سلول پرس‌وجوی پیشوندی تبدیل و با استفاده از پاسخ هر سلول به پاسخ نهایی برسیم.

درخت تجمعی پیشوندی، از بین شرایط زیر، برای مورد آخر راه‌حلی ارائه نکرده و آن را نادیده گرفته است.

- عدم ذخیره‌سازی مقادیر تهی
- امکان محاسبه پاسخ‌های ذخیره‌نشده از روی پاسخ‌های ذخیره‌شده
- احتمال استفاده از داده‌های ذخیره‌نشده،

در روش پیشنهادی سعی شده با افزایش احتمال استفاده از داده‌های ذخیره‌شده، برخط‌بودن را افزایش دهیم.

۳-۱- یافتن درخت پیشوندی بیشینه

اگر ترتیب ابعاد درخت را به‌گونه‌ای انتخاب کنیم که عمده درخواست‌ها از نوع پیشوندی یا میانوندی باشد، پاسخ‌ها در ساختار درخت وجود داشته و برخط‌بودن را افزایش داده‌ایم. اگر این درخت را یک درخت بیشینه بنامیم، هدف ما تشخیص و یافتن این درخت است.

یک مجموعه سلول پرس‌وجو به نام C را در نظر بگیرید، در این مجموعه برای هر بعد تعداد رؤیت مقادیر * و غیر* را می‌توان شمارش کرد. (رؤیت * در یک بعد به معنی عدم درخواست روی آن بعد است) اگر ابعاد را به‌ترتیب نزولی تعداد رؤیت مقادیر غیر* آن‌ها مرتب کنیم، در این صورت ابعادی که بیشترین درخواست روی آن‌ها بوده در ابتدا و به‌ترتیب، ابعاد با درخواست کمتر در ادامه ظاهر شده‌اند. این ترتیب به‌دست‌آمده، نشان می‌دهد که در مجموعه C به‌طورمعمول درخواست‌ها روی چه ابعادی بیشتر و روی چه ابعاد دیگری کمتر است. اگر این ترتیب را به رفتار سامانه تعبیر کنیم، در این صورت تغییر رفتار سامانه به معنی تغییر این ترتیب و به‌عبارتی تغییر ابعاد پرکاربرد است.

نکته: فرض کنیم که سامانه تغییر رفتار نداشته باشد، در این صورت پیشنهاد درخت پیشوندی که بیشتر سلول‌های تجمعی در آن از نوع پیشوندی و یا میانوندی باشند، امکان‌پذیر خواهد بود.

این مطلب را به بیانی دیگر شرح می‌دهیم. فرض کنید یک مجموعه از سلول‌های پرس‌وجو به نام C داریم؛ برای هر درخت پیشوندی T_i، تعدادی از سلول‌های C، پیشوندی هستند. اگر مجموعه سلول‌های تجمعی پیشوندی را C_{p_i} (بدیهی است که همواره $C_{p_i} \subset C$ برقرار است) بنامیم، T_iای را که در آن کاردینالیتی C_{p_i} بیشینه باشد، درخت بیشینه می‌نامیم.

ازآنجایی‌که با تغییر ترتیب ابعاد، به درخت‌های متفاوتی می‌رسیم، پس اعضای مجموعه C_{p_i} نیز به‌ازای درخت‌های T_i، می‌تواند متفاوت باشد. اگر در مجموعهٔ C،

ابعاد با بیشترین فراوانی را به‌ترتیب در عمق نزدیک‌تری به ریشه، قرار دهیم، تعداد بیشتری از سلول‌های مجموعه C، توانسته‌اند در درخت جدید تشکیل سلول پیشوندی بدهند.

به‌طور خلاصه اگر برای مجموعهٔ C ابعاد را به‌ترتیب نزولی حضور مقادیر غیر*شان مرتب کنیم و درخت را با همین ترتیب بسازیم، به‌دلیل افزایش تعداد سلول‌های تجمعی پیشوندی و ایجاد درخت بیشینه، توانسته‌ایم زمان اجرا را در کل مجموعه C کاهش دهیم.

با توجه به مطالب ارائه‌شده، برای به‌دست‌آوردن درخت جدید بدین ترتیب عمل می‌کنیم: با رسیدن هر سلول پرس‌وجو، یک کار آماری انجام داده و میزان انحراف درخت فعلی را از درخت بیشینه به‌دست می‌آوریم:

- به‌ازای هر بعد، یک متغیر δ_i در نظر گرفته و با کمک آن تعداد حضور مقادیر غیر* را برای آن بعد شمارش کنیم؛ بنابراین مجموعه‌ای به‌شرح $\Delta = \{\delta_i | 0 \le i \le n\}$ خواهیم داشت که در آن n تعداد ابعاد است.
- از آنجایی که در درخت بیشینه δ_iها به‌صورت نزولی مرتب شده‌اند، کافی است مجموعه Δ را به‌ترتیب نزولی مرتب کرده و آن را Δ' بنامیم.
- انحراف دو مجموعه را از رابطه (۱) به‌دست می‌آوریم.

$$\mathbf{f(\Delta) = \sum_{i=1}^{n} (\delta_i - \delta'_i)^2} \qquad (۱)$$

نکته: در یک مجموعه نزولی، همواره $\delta_i - \delta'_i$ برابر صفر و درنتیجه حاصل انحراف $f(\Delta)$ صفر است. (یعنی در حالتی که مجموعه نزولی باشد، انحرافی نداریم.)

از آنجایی که در مدت زمان طولانی مقدار δ_i رو به افزایش است، پس فضای مورد نیاز برای نگهداری آن‌ها نیز باید نامحدود باشد که درعمل غیرممکن است. بنابراین، δ_iها را به‌صورت یک مقدار درصدی نگهداری می‌کنیم.

$$\mathbf{inc(\delta_i) = \frac{\delta_i + 1}{101} \times 100}$$
$$\mathbf{dec(\delta_i) = \frac{\delta_i}{101} \times 100} \qquad (۲)$$

نکته: درصورتی‌که برای δ_iها مقدار غیر صفر را به‌عنوان مقدار اولیه در نظر بگیریم، δ_iها همواره مخالف صفر خواهند بود. همچنین به‌منظور مقایسه ساده‌تر؛ $f(\Delta)$ را نیز به‌صورت درصدی به‌دست می‌آوریم.

$$f(\Delta) = \frac{\sum_{i=1}^{n} (\delta_i - \delta'_i)^2}{Max(f)} \times 100 \qquad (۳)$$

لم ۱۰-: بیشترین مقدار $\sum_{i=1}^{n} (\delta_i - \delta'_i)^2$ برابر $n \times 10^4$ خواهد بود.

اثبات: مشخص است که بیشترین اختلاف دو عدد در بازه [0,100]، ۱۰۰ خواهد بود. با این توضیح بیشترین مقدار (Δ) به شرح ذیل است:

$$\mathbf{Max\left(\sum_{i=1}^{n} (\delta_i - \delta'_i)^2\right)} = \mathbf{\sum_{i=1}^{n} \left(Max(\delta_i - \delta'_i)\right)^2} = \mathbf{\sum_{i=1}^{n} 100^2 = n10^4} \qquad (۴)$$

از مقایسه شاخص به‌دست آمده $f(\Delta)$ با یک مقدار ثابت مانند φ، می‌توانیم زمان اصلاح درخت را تشخیص دهیم (هرگاه $f(\Delta) > \varphi$ برقرار بود، درخت با ترتیب جدید اصلاح می‌شود). تعیین مقدار φ به تغییر رفتارهای سامانه بستگی دارد.

به‌طورکلی روش پیشنهادی برای حالاتی که تغییر رفتار سامانه کم است بهتر کار می‌کند و استفاده از φ کوچک‌تر به معنای اصلاح هر چه زودتر درخت است. در حالاتی که تغییر رفتار سامانه زیاد است برای خنثی‌کردن روش پیشنهادی از φ بزرگ‌تر استفاده می‌شود. انتخاب φ مناسب بسیار اهمیت دارد. به‌عنوان مثال اگر φ کوچک باشد، ممکن است درخت قبل از تشخیص صحیح ترتیب ابعاد تغییر کند.

برای تشریح بیشتر به مثال ذیل توجه نمایید:

درختی با پنجره لغزان $[1,2]$ و تعداد ۴ بعد در نظر بگیرید. پس $\Delta = \{\delta_1, \delta_2, \delta_3, \delta_4\}$ خواهد بود. تمامی عناصر با مقدار پنجاه درصد مقداردهی اولیه می‌شوند. پس $\Delta = \{50,50,50,50\}$ است.

فرض کنید سلول‌های پرس‌وجوی مجموعه C، رسیده باشند.

$$\mathbf{C = \{* b_1 **, a_2 b_1 * d_1, ** c_1 d_2, ****, * b_2 * d_1, a_3 * **\}}$$

به نخستین پرس‌وجو $b_1 * **$ دقت کنید. در بعد نخست مقدار * آمده است، پس باید δ_1 را کاهش دهیم. در

بعد دوم مقدار غیر * داریم و لازم است δ_2 را افزایش دهیم. همین‌طور ابعاد سوم و چهارم نیز باید کاهش یابند. محاسبات Δ برای سلول‌های پرس‌وجوی داده‌شده در جـــــدول(۱) آمده است.

(جدول-۱): یک مثال از محاسبات Δ

(Table-1): An example of Δ calculations

ردیف	سلول پرس‌وجو	محاسبات Δ	$f(\Delta)$
۱	$* \mathbf{b_1} **$	$\delta_1 = 49.5$ $\delta_2 = 50.49$ $\delta_3 = 49.5$ $\delta_4 = 49.5$	**0.0049**
۲	$\mathbf{a_2 b_1} * \mathbf{d_1}$	$\delta_1 = 50$ $\delta_2 = 50.98$ $\delta_3 = 49$ $\delta_4 = 50$	**0.0098**
۳	$** \mathbf{c_1 d_2}$	$\delta_1 = 49.5$ $\delta_2 = 50.47$ $\delta_3 = 49.5$ $\delta_4 = 50.49$	**0.0049**
۴	****	$\delta_1 = 49$ $\delta_2 = 49.97$ $\delta_3 = 49$ $\delta_4 = 49.99$	**0.0049**
۵	$* \mathbf{b_2} * \mathbf{d_1}$	$\delta_1 = 48.51$ $\delta_2 = 59.38$ $\delta_3 = 48.51$ $\delta_4 = 59.4$	**0.59**
۶	$\mathbf{a_3} ***$	$\delta_1 = 49.02$ $\delta_2 = 58.79$ $\delta_3 = 48.03$ $\delta_4 = 58.81$	**0.53**

در هر مرحله، پس از تکمیل محاسبات Δ به محاسبه انحراف از درخت بیشینه می‌پردازیم تا درصورت رسیدن به مقدار ثابت φ نسبت به اصلاح درخت اقدام کنیم. براشکلی ردیف ۱ جدول(۱) محاسبه $f(\Delta)$ را نشان می‌دهیم.

ابتدا لازم است مجموعه Δ' را به‌دست آوریم:

$$\left.\begin{matrix}\Delta' = \{50.49, 49.5, 49.5, 49.5\} \\ \Delta = \{49.5, 50.49, 49.5, 49.5\}\end{matrix}\right\} \Rightarrow \mathbf{f(\Delta)} = \frac{(50.49-49.5)^2 + (49.5-50.49)^2 + 0 + 0}{40000} = \frac{(0.99)^2 + (0.99)^2}{40000} = \frac{1.96}{40000} \quad (۵)$$

محاسبه $f(\Delta)$ به‌صورت کامل در جدول (۱) آورده شده است. بدین ترتیب اگر $\varphi = 20\%$ در نظر گرفته شود، درخت در هیچ‌یک از مراحل مثال آورده‌شده، نیازی به تغییر ندارد.

۲-۳- اصلاح درخت و پاسخ به پرس‌وجو

همان‌طور که گفته شد، با استفاده از مقایسه $f(\Delta)$ با φ می‌توانیم زمان اصلاح درخت را تشخیص دهیم. برای این منظور، دو روش پیش رو داریم:

۱-۲-۳- تغییر ترتیب ابعاد درخت

درصورتی‌که تصمیم گرفتیم درخت را تغییر دهیم، یک روش ساده‌لوحانه تبدیل درخت است. در این روش درصورتی‌که هزینه تبدیل را بپذیریم، در پاسخ‌گویی به پرس‌وجوها هزینه جدیدی را تحمل نکرده‌ایم؛ ولی با توجه به اینکه این تبدیل، مستلزم مسدودکردن جریان بوده ما را از برخط‌بودن دور می‌کند. بنابراین از این روش صرف‌نظر می‌شود.

ایجاد یک درخت جدید

در این روش دو درخت داریم؛ یکی را فعال[1] و دیگری را منقضی[2] می‌نامیم. در شروع فقط درخت فعال وجود دارد؛ پس از مدتی و با استفاده از محاسبات $f(\Delta)$ نتیجه می‌گیریم که درخت جدیدی ایجاد کنیم؛ درخت فعلی را منقضی و درخت جدید را با عنوان فعال تشکیل می‌دهیم.

نکته: اصلاحات حاصل از محاسبات $f(\Delta)$ به شرطی انجام می‌شود که تنها یک درخت وجود داشته باشد. حال الگوریتم ساخت و نگهداری درخت که درقبل روی یک درخت اجرا می‌شد، باید برای هر دو درخت اجرا شود.

- **عمل درج تاپل جدید:** تاپل جدید همواره به درخت فعال اضافه می‌شود و هیچ تفاوتی با روش درخت ایستا ندارد.
- **عمل حذف سطرهای با مهر زمانی قدیمی:** پس از درج تاپل جدید، تعدادی از گره‌های جدول فعال به‌عنوان یک اثر جانبی درج، اصلاح می‌شوند. با استفاده از متغیر LUT از برگ‌ها به سمت ریشه حرکت می‌کنیم و گره‌هایی را که در جداول تجمعی آن‌ها، تنها اطلاعات مربوط به مهر زمانی قدیم وجود دارد، حذف می‌کنیم. درصورتی‌که تمام سطرهای جدول تجمعی حذف شد، خود گره را نیز حذف می‌کنیم. لازم به ذکر است که عمل حذف، برخلاف عمل درج بر روی هر دو درخت انجام می‌شود.
- **عمل به‌روزرسانی جداول تجمّعی:** به‌روزرسانی، برای جداول تجمعی هر گره و یال‌های میانوندی

[1] Active

[2] Expired

انجام می‌شود. در زمان درج یک تاپل جدید عمل به‌روز رسانی نیز انجام می‌شود. چون عمل درج تنها مربوط به درخت فعال است، پس عمل به‌روزرسانی، تنها برای درخت فعال انجام می‌شود؛ بدین ترتیب درخت‌های فعال و منقضی به‌گونه‌ای ساخته و نگهداری می‌شوند که در آن رفته‌رفته درخت منقضی از بین رفته و درخت فعال باقی می‌ماند. الگوریتم (۱)، ساخت و نگهداری درخت را به‌صورت پویا شرح می‌دهد.

۲-۲-۳- پاسخ به سلول‌های پرس‌وجو

برای پاسخ به سلول‌های پرس‌وجو در روش درخت ایستا، تنها یک درخت وجود داشت و الگوریتم تنها روی این درخت اجرا می‌شد؛ ولی در روش جدید دو درخت داریم که عمل به‌روزرسانی جداول تجمعی هیچ‌کدام از درخت‌ها به‌صورت کامل اجرا نشده است؛ لذا ضروری است که الگوریتم پاسخ به پرس‌وجو، روی هر دو درخت اجرا و با استفاده از نتایج به‌دست‌آمده، پاسخ نهایی نیز محاسبه شود. الگوریتم (۲)، پاسخ به پرس‌وجو را برای درخت پویا شرح می‌دهد. در این الگوریتم، عبارات phi و f_{delta} به‌ترتیب همان φ و $f(\Delta)$ هستند که پیش‌تر توضیح داده شده‌اند.

در بخش بعدی نشان خواهیم داد که تغییرات اعمال‌شده، زمان اجرای پاسخ به پرس‌وجو را در ازای افزایش قابل قبول فضای حافظه مصرفی، بهبود بخشیده است.

(الگوریتم-۱): ساخت و نگهداری درخت پویا
(Algorithm-1): Dynamic Tree construction

```
Algorithm: Dynamic Tree construction
Input: the current sliding window
Output: a PAT
Method:
    initialize a tree with only the root as ActiveTree;
    initialize phi with a default value
    read the tuple into main memory one by one, one at each time;
    FOR each tuple DO
        insert_tupple (tuple, ActiveTree);
        removeOldInsatnce (ActiveTree);
        IF exist ExpireTree THEN
            removeOldInsatnce (ExpireTree);
Function insert_tupple (current-tuple, tree)
    FOR each node on the path of the current-tuple DO
        update the aggregate at the node;
        if it is a new node, adjust infix links and side-links
        IF v's aggregate table contains a row of instant at instant (t- ω +1), THEN
            remove the row from the aggregate table;
        IF v is a leaf node THEN put v into the LUT list of LUT=t+1;
Function removeOldInsatnce (tree)
    FOR each node v in the LUT list of the tree that LUT=t- ω +1 DO
        search v's ancestors upward until a node having aggregate after instant t- ω+1 is encountered;
        remove v and those ancestors of v and the infix links,adjust the related side-links, too
```

(الگوریتم-۲): پاسخ به پرس‌وجو
(Algorithm-2): Answering queries

```
Algorithm: Answering queries
Input: the PATs at instant t and a query cell (τ, d1, ... , dn)
Output: the aggregate of the query cell
Method:
    A1= Answer (ActiveTree,q);
    A2= Answer (ExpireTree,q);
```

```
    RETURN (aggr(A1,A2));

Function Answer (tree, q-cell)
    f_delta= Calculate_f_delta(d1, ... , dn);
    IF f_delta of ActiveTree >= phi AND ExpireTree do not exits THEN
            change ActiveTree to ExpireTree;
            initialize a tree with only the root as ActiveTree;
    IF τ≠* and the root's aggregate table does not have a bout τ THEN RETURN (null);
    let current-node=root of tree;
    m=search (current-node , q);
    RETURN (m);

Function search (current-node, q-cell)
  suppose q − cell = (τ, d1, ... , dl);
  IF d1≠* THEN
    IF current-node has a child v of label d1 THEN
      IF τ≠* and there is no row of instant τ in the aggregate table v THEN RETURN (null);
      ELSE IF l=1 THEN RETURN the aggregate table v
          ELSE current-node = v; q' = (τ, d2, ... , dl); m =search (current-node , q');RETURN (m);
    ELSE RETURN (null);
  ELSE //case d1 = *
    IF d1 = ⋯ = dl = ∗ THEN RETURN the aggregate at current-node;
    let di be the first dimension non-* value in q;
    IF current-node has no infix link of label di THEN RETURN (null);
    IF τ≠* and the aggregate table of the infix link of di has no row of τ THEN RETURN (null);
    IF di be the only non-* value in q THEN RETURN the aggregate at the infix link of di;
    m = 0;
    follow the side links, visit every node carrying label di in the subtree of current-node,
    FOR each node v in the linked list DO
      let current-node = v; q' = (di+1, ... , dl); m = aggr (m,search (current-node,q'))
    RETURN (m)

Function Calculate_f_delta(dimensions)
  Calculate the value of f_delta according to section 3-2
```

(جدول-۲): مرتبه زمان اجرای دستورات مختلف در الگوریتم پاسخ به پرس‌وجوی روش درخت پیشوندی ایستا

(Table-2): The runtime order of different commands in query answering algorithm using static prefix aggregate tree

No#	**Algorithm:** Answering point queries	T
1	**Input**: the PAT *T* at instant *t* and a query cell (*τ, d1, ... , dn*)	
2	**Output**: the aggregate of the query cell	
3	**Method**:	
4	IF τ≠* and the root's aggregate table does not have a bout *τ* THEN RETURN (*null*);	$O(1)$
5	let *current-node=root*;	
6	*m*=search (*current-node* , *q*);	$O(T_n)$
7	RETURN (*m*);	
8	**Function** search (current-node, q-cell)	
9	suppose $q - cell = (\tau, d_1, \dots, d_l)$;	
10	IF $d_1 \neq *$ THEN	
11	IF current-node has a child *v* of label d_1 THEN	
12	IF τ≠* and there is no row of instant *τ* in the aggregate table *v* THEN RETURN (*null*);	$O(1)$
13	ELSE IF *l= 1* THEN RETURN the aggregate table *v*	$O(1)$
14	ELSE current-node = *v*; $q' = (\tau, d_2, \dots, d_l)$; *m* =search (*current-node* , *q'*);RETURN (*m*);	$O(T_{n-1})$
15	ELSE RETURN (*null*);	$O(1)$
16	ELSE //case d_1 = *	
17	IF $d_1 = \cdots = d_l = *$ THEN RETURN the aggregate at *current-node;*	$O(1)$
18	let d_i be the first dimension non-* value in *q*;	
19	IF *current-node* has no infix link of label d_i THEN RETURN (*null*);	$O(1)$
20	IF τ≠* and the aggregate table of the infix link of d_i has no row of *τ* THEN RETURN (*null*);	$O(1)$
21	IF d_i be the only non-* value in *q* THEN RETURN the aggregate at the infix link of d_i;	$O(1)$
22	*m = 0*;	
23	follow the side links, visit every node carrying label d_i in the subtree of current-node,	
24	FOR each node *v* in the linked list DO	
25	let current-node = *v*; *q'* = (*di+1, ... , dl*); *m = aggr* (*m*,search (*current-node,q'*)) *RETURN (m)*	$O(1 + l^{i-1}T_{n-i})$

۴- ارزیابی روش پیشنهادی

در بخش قبل نشان دادیم با استفاده از تابع $f(\Delta)$ و مجموعهٔ $\Delta = \{\delta_i | 0 \le i \le n, 0 \le \delta_i \le 100\}$ و مقایسه مقدار این تابع با یک مقدار ثابت مانند φ، می‌توانیم زمان تغییر درخت پیشوندی را تشخیص دهیم. در این زمان درخت جدیدی در کنار درخت قبلی تشکیل می‌دهیم. درخت قدیمی نیز به‌تدریج از بین رفته و درخت جدید جایگزین درخت قدیمی می‌شود. در ادامه به تحلیل فضا و زمان مورد نیاز روش پیشنهادی می‌پردازیم.

روش پیشنهادی با استفاده از درخت بیشینه، سعی دارد زمان پاسخ به پرس‌وجوها را کاهش دهد. در روش پیشنهادی، افزایش حافظه مصرفی به‌دلیل وجود دو درخت هم‌زمان، بدیهی است. در ادامه سعی شده که میزان افزایش حافظه مصرفی محاسبه و بررسی شود که آیا این افزایش مقرون‌به‌صرفه است یا نه. از طرفی هنوز این نگرانی و سؤال وجود دارد که آیا با وجود دو درخت هم‌زمان، روش پیشنهادی می‌تواند زمان اجرا را کاهش دهد؟

هر بار که تاپل جدیدی اضافه می‌شود، تعدادی گره به درخت جدید اضافه و تعدادی گره از درخت قدیم حذف می‌شود. از زمان تشکیل درخت جدید، درخت قدیم هیچ افزایشی نداشته و فقط کاهش گره دارد. درخت جدید در شروع اندازه کوچکی دارد و در ادامه از نظر اندازه جای خود را با درخت قدیم تعویض می‌کند. با این توضیح اندازه حافظه مورد نیاز با توجه به وجود دو درخت، کمتر از دو برابر شده است؛ یعنی حافظه مورد نیاز در بدترین حالت هم سقف دارد و از این‌رو قابل قبول است.

برای محاسبه زمان اجرا، جدول (۲) را به‌دست می‌آوریم. به محل‌های شامل دستور Return دقت شود. ردیف هایی مانند ۱۲،۴ و ۱۳ زمان اجرا از مرتبه $O(1)$ است. ردیف ۶ از مرتبه $O(T_n)$ و ردیف ۱۴ از مرتبه $O(T_{n-1})$ است. (n در آن تعداد ابعاد است) ردیف ۲۵ از مرتبه $O((1+l)^{i-1} \times T_{n-i})$ است. زیرا در محل iام تعداد فهرست کناری حداکثر در بدترین حالت l^{i-1} بوده که به‌ازای هر کدام زمان $O(T_{n-i})$ است.

همان طور که مشخص است با توجه به ورودی (سلول پرس‌وجو)، الگوریتم از مسیرهای مختلفی اجرا و درنتیجه زمان اجرا متفاوت خواهد بود. بهترین زمان اجرا به‌ازای ردیف ۱۴ و بدترین زمان اجرا به‌ازای ردیف ۲۵ حاصل می‌شود. به‌عبارتی بهترین و بدترین زمان اجرا در این الگوریتم به‌ترتیب از مرتبه‌های $O(T_n)$ و $O(l^n)$ خواهد بود.

با استفاده از درخت پویا، همیشه سعی بر این است که ترتیب ابعاد، نزولی باشد. بدین ترتیب احتمال شرکت بعدهای ابتدایی در سلول پرس‌وجو بیشتر و درنتیجه احتمال تشکیلسلول‌های پرس‌وجوی پیشوندی و میانوندی افزایش خواهد یافت.

بدین ترتیب با استفاده از روش درخت پویا، زمان اجرای الگوریتم پاسخ به پرس‌وجو از مرتبه‌های نمایی به مرتبه‌های چندجمله‌ای میل کرده است. حال نوبت به محاسبه زمان اجرای الگوریتم (۱) است. زمان اجرای الگوریتم‌های ساخت و نگهداری درخت ایستا، از مرتبه $O(nl)$ [5] محاسبه شده است. از آنجایی که دو درخت داریم، باید الگوریتم برای هر دو نیز اجرا شود. درضمن برای درخت قدیمی تنها بخش مربوط به حذف الگوریتم اجرا می‌شود. پس زمان اجرا در بدترین حالت کمتر از دو برابر خواهد شد (زمان اجرای بخش حذف‌کردن گره‌های مربوط به پنجره قبلی کمتر از $O(nl)$ محاسبه می‌شود). به عبارت دیگر زمان اجرا هنوز از مرتبه $O(nl)$ است. علاوه بر نتایج بالا، آزمایش‌های صورت‌گرفته نشان داده که روش پیشنهادی عملکرد بهتری نسبت به روش ایستا دارد.

۴-۱- تنظیم‌های آزمایش‌ها

در این بخش نتایج تجربی حاصل از اجرای الگوریتم‌های ایستا و پویا ارائه می‌شود. کلیه الگوریتم‌ها در محیط VS 2008 و زبان C# پیاده‌سازی و در سیستمی با مشخصات زیر اجرا شده است:

- سیستم‌عامل: ویندوز ۷، ۳۲ بیتی
- پردازنده: Pentium(R) Dual-Core CPU E5700 @ 3.00GHz 3.00 GHz
- حافظه اصلی: ۳ گیگابایت

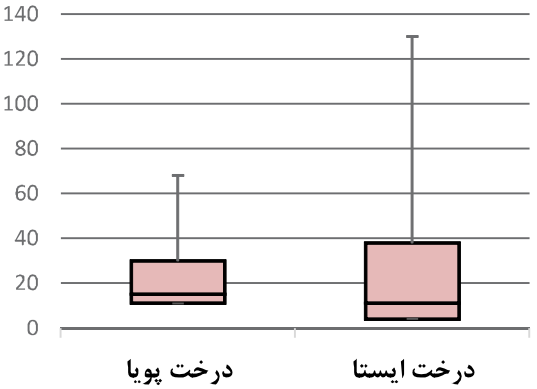
جهت ارزیابی روش پیشنهادی از اطلاعات واقعی هواشناسی (حدود چهارده میلیون رکورد) در منبع [6] شامل ۱۲۲ ایستگاه هواشناسی بین سال‌های ۱۹۹۱ تا ۲۰۱۰ استفاده شده است. مجموعه داده‌های هواشناسی برای محاسبات مکعب داده معیارهای مناسبی هستند [7]، [8]، [9] و [10]. درضمن آزمایش‌های انجام‌شده در [5] نیز روی همین بستر داده اجرا شده است.

به‌منظور ایجاد سلول پرس‌وجو غیر تهی، ده درصد از مجموعه داده‌ها به‌صورت تصادفی انتخاب و از روی آن تعداد

هزار سلول پرس‌وجو به‌صورت تصادفی ایجاد کرده‌ایم. الگوریتم‌های ایجاد درخت پیشوندی و پاسخ به پرس‌وجو در هر دو روش ایستا و پویا، پیاده‌سازی و روی این مجموعه اجرا شده است. برای هر روش، اندازه پنجره‌های لغزان، سیصد، ششصد و نهصد در نظر گرفته شده است.

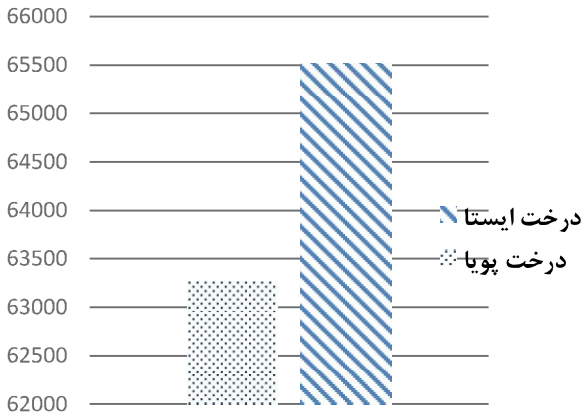
۲-۴- ارزیابی پاسخ به پرس‌وجو

با توجه به اینکه روش پویا برای تمامی پرس‌وجوها نمی‌تواند بهینه باشد، لذا دو روش را در مهرهای زمانی مختلف مقایسه نمی‌کنیم؛ بلکه نشان می‌دهیم روش پویا در مجموع عملکرد بهتری دارد. همچنین به‌منظور مقایسه بهتر و دقیق‌تر، زمان اجرای الگوریتم، برحسب تعداد دستورهای اجراشده ارزیابی شده است. نمودارهای شکل‌های (۲) و (۳)، نشان می‌دهند که تعداد دستورهای اجراشده (به عبارت دیگر، زمان اجرای الگوریتم پاسخ به پرس‌وجو) در روش پویا نسبت به روش ایستا کمتر است.



(شکل-۲): مقایسه تعداد دستورات اجراشده با استفاده از نمودار جعبه‌ای

(Figure-2): The number of instructions executed by dynamic tree v.s static

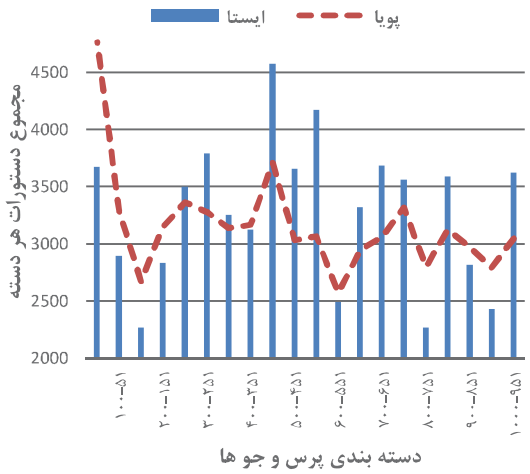


(شکل-۳): مجموع تعداد دستورات اجرا شده در درخت‌های پویا و ایستا

(Figure-3): The total number of instruction executed by dynamic tree v.s static tree

نمودار شکل (۲) نشان می‌دهد زمان اجرای الگوریتم پاسخ به پرس و جوی روش پویا از نظر شاخص‌های مرکزی و پراکندگی نسبت به روش ایستا برتری دارد. نمودار شکل (۳) نیز بیان‌گر این است که مجموع تمامی دستورهای روش پویا نیز از روش ایستا کمتر است.

نمودار شکل (۴) را در نظر بگیرید. محور افقی این نمودار، دسته‌های پنجاه‌تایی سلول‌های پرس‌وجو برای هر پنجره لغزان (۱۵۰ پرس‌وجو) و محور عمودی مجموع تعداد دستورات هر دسته را نمایش می‌دهد. این نشان می‌دهد که در پرس‌وجوهای ابتدایی زمان اجرای روش پویا بیشتر است. با گذشت زمان و رسیدن پرس‌وجوهای بعدی و تغییرات ساختار درختی، در اکثر موارد زمان اجرا کاهش یافته است.



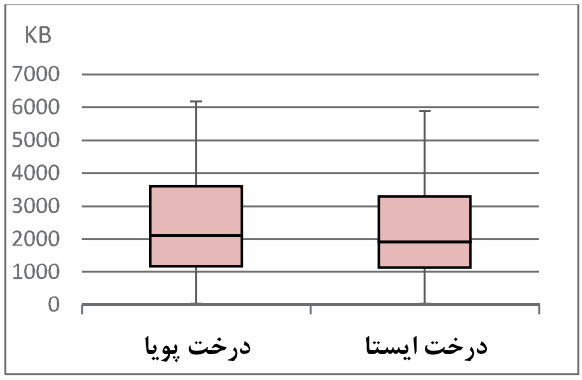
(شکل-۴): تعداد دستورات اجرا شده‌ی هر پرس‌وجو

(Figure-4): The total number of instructions executed by dynamic tree v.s static tree

۳-۴- ارزیابی ساخت و نگهداری درخت

آنچه مسلم است روش پویا به حافظه بیشتری نیاز دارد و با وجود دو درخت میزان حافظه مصرفی باید به احتمال زیاد دو برابر باشد؛ ولی آزمایش انجام‌شده، نشان داد که این نسبت بسیار کمتر از دو است. نمودار شکل (۵) نشان می‌دهد که با وجود اینکه روش پویا به حافظهٔ بیش‌تری نیاز داشته، ولی شاخص‌های پراکندگی و مرکزی بسیار نزدیکی به روش ایستا دارد.

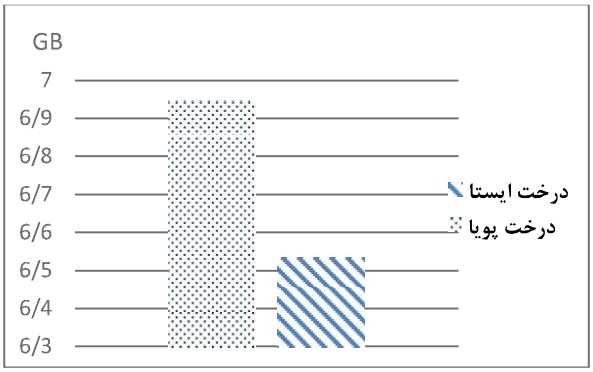
نمودار شکل (۶) مجموع اندازه مصرفی درخت پویا و درخت ایستا را مقایسه کرده است. این مقایسه نشان می‌دهد که نسبت میزان حافظه مصرفی روش پویا به ایستا بسیار کمتر از دو هست.



(شکل-۵): مقایسه حافظهٔ مصرفی با استفاده از نمودار جعبه‌ای

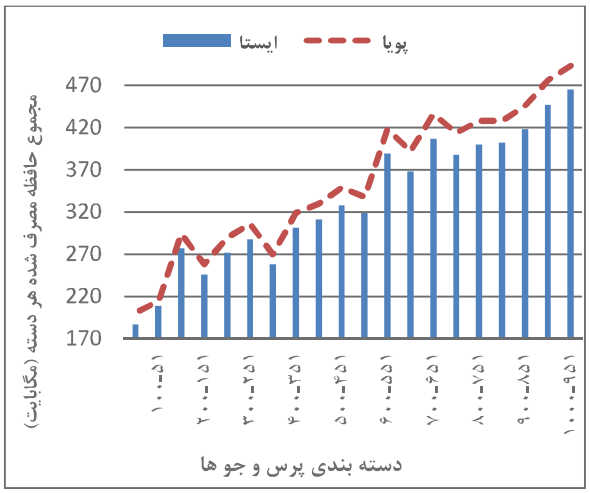
(Figure-5): Box-plots comparing the size of dynamic tree v.s static tree

نمودار شکل (۷) نیز سلول‌های پرس‌وجو را دسته‌بندی و میزان حافظهٔ مصرفی در هر دسته را نشان می‌دهد. این نمودار نشان می‌دهد که میزان افزایش حافظهٔ مصرفی فاصله نزدیکی نسب به روش ایستا دارد.



(شکل-۶): مجموع فضای مصرف‌شده در درخت‌های پویا و ایستا

(Figure-6): The total memory used by dynamic tree v.s static tree



(شکل-۷): فضای مصرف‌شدهٔ هر پرس و جو

(Figure-7): The total memory used by dynamic tree v.s static tree

از آزمایش‌های صورت‌گرفته می‌توان نتیجه گرفت:

- زمان اجرای روش پیشنهادی در کل کمتر از روش ایستا است.
- میزان افزایش یافته نسبت به روش ایستا بسیار کمتر از دو برابر است.

این نتایج بیان می‌کند که روش پیشنهادی با افزایش اندک فضای حافظه، عملکرد بهتری نسبت به روش ایستا دارد. از طرفی با توجه به ارزیابی‌های صورت‌گرفته در [5] و اثبات کارایی بهتر آن نسبت به سایر روش‌های موجود، می‌توان نتیجه گرفت که روش پیشنهادی کارایی بهتری نسبت به سایر روش‌هایی که در ادامه آورده شده، دارد. ازاین‌رو از ارائهٔ آزمایش‌ها و مقایسه‌های بیشتر صرف‌نظر شده است.

۵- کارهای مرتبط

در این بخش با تعدادی از کارهایی مرتبط با پاسخ به پرس‌وجوهای تجمعی روی جریان‌های داده آشنا می‌شویم.

۵-۱- توابع تجمعی پنجره‌ای[1]

به‌منظور کاهش زمان اجرا و فضای مورد نیاز برای یک پرس‌وجوی جریان داده، می‌توان پنجره‌هایی را که هم‌پوشانی دارند، به تعدادی قاب[2]های مجزا تقسیم کرد؛ سپس روی هر قاب یک تجمع جزئی را حساب کرده و درنهایت این محاسبات را برای هر پنجره تعمیم داد [11].

این روش با استفاده از محاسبات تجمعی جزئی[3] به کاهش اندازه بافر و با استفاده از به اشتراک‌گذاری آن‌ها به کاهش هزینه محاسبات می‌پردازد. این رویکرد روی بخش‌هایی از جریان که به آن قاب گفته می‌شود، تجمع جزئی را حساب می‌کند. این قاب‌ها در شکل (۸) نشان داده شده‌اند. هر جریان به قاب‌های یک‌دقیقه‌ای، تقسیم می‌شود. بدین ترتیب در هر جریان چهار دقیقه‌ای چهار قاب وجود دارد. شکل (۸)، پنجره W3 شامل قاب‌های P3 تا P6 را نشان می‌دهد. هر قاب هم در چهار پنجره شرکت دارد، به‌عنوان‌مثال P5 در پنجره‌های W2 تا W5 شرکت دارد. برای تکمیل محاسبه پرس‌وجو تجمع قاب‌های یک پنجره محاسبه می‌شود.

[1] Window Aggregate Queries

[2] Pane

[3] sub-aggregating

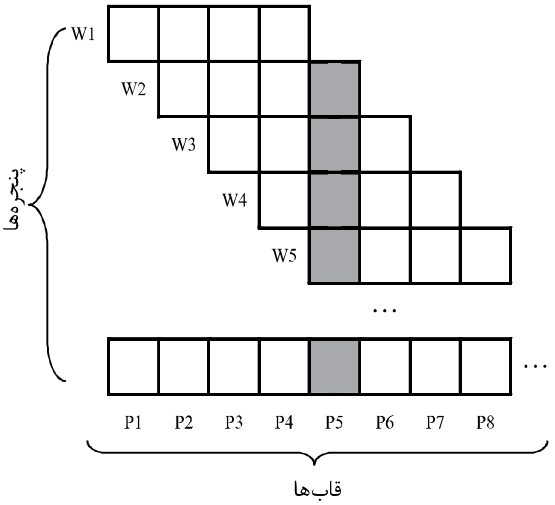
۲-۵- پـــــردازش پرس‌وجـــــوی تجمعـــــی درون‌شبکه‌ای[1]

یک رویکرد ساده برای ارزیابی یک پردازش پرس‌وجوی تجمعی، هدایت تمامی مقادیر حس‌شده به ایستگاه پایه[2] و محاسبه مقدار تجمعی در آنجا ایستگاه هست. هرچند که این رویکرد ساده است، ولی تعداد پیام‌ها و مصرف انرژی می‌تواند زیاد باشد. روش بهتر، استفاده از توان محاسباتی تجهیزات حسگر و محاسبات تجمعی درون‌شبکه‌ای است. همه توابع تجزیه‌شدنی را می‌توان به‌صورت توابع تجمعی درون‌شبکه‌ای محاسبه کرد [12]. تابع f تجزیه‌شدنی است، درصورتی‌که بتواند با توابع دیگر g به‌صورت (۶) محاسبه شود [13]:

$$\mathbf{f}(\mathbf{v_1}, \mathbf{v_2}, \cdots, \mathbf{v_n}) = \mathbf{g}(\mathbf{f}(\mathbf{v_1}, \cdots, \mathbf{v_k}), \mathbf{f}(\mathbf{v_{k+1}}, \cdots, \mathbf{v_n})) \quad (۶)$$

با استفاده از توابع تجزیه‌شدنی، مقدار تابع تجمعی[3] برای زیرمجموعه‌های گسسته[4] می‌تواند محاسبه برای محاسبه تجمع کل مجموعه با استفاده از تابع ادغام g به‌کار روند. این روش در چارچوب TAG[5] برای TinyDB به‌کار رفته است [12]. علاوه‌براین روش‌های مشابه دیگری برای محاسبه توابع تجمعی وجود دارد [14]، [15]، [16]و [17].

پردازش پرس‌وجوی درون‌شبکه‌ای در TAG، شامل دو مرحلهٔ توزیع[6] و جمع‌آوری[7] است. در حین مرحلهٔ توزیع، پردازش پرس‌وجو در شبکه منتشرشده و گره‌ها را به‌صورت یک درخت تجمعی[8] سازمان‌دهی می‌کند. ایستگاه پایه که پردازش پرس‌وجو را به همه گره‌ها منتشر می‌کند[9] ، ریشه درخت است. در پیام پردازش پرس‌وجو یک شمارنده وجود دارد، که در هر ارسال مجددی افزایش یافته که همان تعداد فاصله هاپ[10] از ریشه است. در این حالت، هر گره به سطحی خاص اختصاص پیدا می‌کند که با فاصلهٔ هاپ گره از ریشه برابر است. همچنین، هر حس‌گری یکی از همسایگانش را که فاصلهٔ کمتری نسبت به فاصله هاپ از ریشه دارد؛ به‌عنوان گره پدر در درخت تجمعی انتخاب می‌کند.



(شکل-۸): پنجره‌ها ترکیبی از چهار قاب هستند
(Figure-8): Windows containing four panes

در حین مرحلهٔ جمع‌آوری، هر گره برگ، یک تاپل را ایجاد و به پدرش ارسال می‌کند. گره‌های غیر برگ تاپل‌های فرزندانشان را دریافت و با یکدیگر ترکیب می‌کنند؛ سپس، این نتایج جزئی جدید را به پدرانشان تسلیم می‌کنند. این فرایند به‌طور مداومی انجام تا بعد از گذشتن h (ارتفاع درخت تجمعی) مرحله نتیجه کلی به ریشه برسد. برای ذخیره بیشتر انرژی، تا جایی که ممکن باشد، گره‌های حسگر در هر مرحله به خواب می‌روند. به‌عبارتی پردازنده و رادیوی آن‌ها بی‌کار هستند. هنگامی‌که زمان تعریف‌شده به پایان برسد یا یک رویداد خارجی واقع شود، دستگاه از خواب بیدار شده و مراحل پردازش و ارتباطی را آغاز می‌کند. در این لحظه، بعد از دریافت پیام‌ها از فرزندانش، مقدار/های جدید را به پدرش تسلیم می‌کند. در پایان، اگر پردازش دیگری لازم نباشد، دوباره به خواب می‌رود [18].

راه‌کار جایگزین دیگر، استفاده از حافظه نهان در شبکه و گره‌های آن است. روش‌های گوناگونی برای بهره‌گیری از حافظه نهان و به‌خصوص حافظه نهان معنایی ارائه شده‌اند، که علاوه‌بر مقدار داده، معنا و کاربرد آن را هم نگه می‌دارند و عملکرد مؤثرتر خواهند داشت [19] و [20].

۳-۵- اسکچ‌های شمارشی[11]

اسکچ‌های شمارشی توسط Flajolet و Martin [21]، به‌منظور تخمین سریع تعداد اقلام مجزا در یک پایگاه داده (یا جریان

[1] In-network Aggregate Query Processing
[2] Base station
[3] Aggregate function
[4] Disjoint
[5] Tiny Aggregation (TAG)framework
[6] Distribution phase
[7] Collection phase
[8] Aggregation tree
[9] broadcast
[10] hop distance
[11] Counting Sketches

داده) به‌صورت اجرای تک‌مسیره[1] و تنها با استفاده از یک فضای حافظهٔ کوچک، معرفی شد. از آن زمان به بعد، اقدامات بسیار زیادی [22]، [23]، [24]، [25]، [26] و [27] برای توسعه و عمومی‌سازی اسکچ‌های شمارشی صورت گرفت. برای شمارش دقیق اعضای متمایز[2]، به فضای $\Omega(n)$ ولی برای شمارش تقریبی آن در یک multi-set[3] با n عضو متمایز، به فضای $\Theta(\log n)$ نیاز داریم [22].

حال اسکچ‌های FM را برای مسئله شمارش اعضای متمایز شرح می‌دهیم. یک multi-set با اعضای $M=\{x_1,x_2,x_3,\ldots\}$ را در نظر بگیرید. مسئله شمارش از طریق محاسبه عبارت زیر صورت می‌گیرد:

$$\mathbf{n \equiv |distint\ (M)|} \qquad (۷)$$

اگر M، یک multi-set باشد، در روش FM، طرح اسکچ آرایه‌ای از بیت‌ها[4] به طول K بوده و به‌صورت S(M) نمایش داده می‌شود. اعضای این آرایه را با صفر مقداردهی اولیه می‌کنیم. این اعضا با استفاده از یک تابع hash دودویی تصادفی[5] h که روی عناصر M اعمال می‌شود، یک می‌شوند. به‌عبارتی:

$$\mathbf{S\ (M)\ [i]\ \equiv\ 1\ if\ \exists x \in M\ s,t,\ min\ \{j|\ h\ (x,j) = 1\} = i} \qquad (۸)$$

با این تعریف، اگر xای وجود داشته باشد که به‌ازای کمترین i، $h(x,i)=1$ شود، می‌تواند تنها یک بیت از S(M) را یک کند. بدین ترتیب یک پیاده‌سازی سریال ساده داریم که درعمل بسیار سریع بوده و به‌طور متوسط برای هر گزینه به دو بار فراخوانی h نیاز دارد. برای درک بهتر، مثال زیر را در نظر بگیرید.

مقداردهی اولیه اسکچ

0	0	0	0	0	0	0	0	0

درج i_1 به درون اسکچ

$\mathbf{h(i_1,1) \rightarrow 0}$

$\mathbf{h(i_1,2) \rightarrow 0}$

$\mathbf{h(i_1,3) \rightarrow 1}$

0	0	1	0	0	0	0	0	0

درج i_2 به درون اسکچ

$\mathbf{h(i_2,3) \rightarrow 1}$

1	0	1	0	0	0	0	0	0

[1] One pass

[2] Distinct counting

[3] یک توسعه از مجموعه است که در آن هر عضو می‌تواند تکرار شود. کاردینالیتی آن برابر تعداد اعضا به همراه تکرارشان و مالتی‌پلیسیتی هر عضو برابر تعداد تکرار آن عضو هست.

[4] Bitmap

[5] random binary hash function

برای یک اسکچ از نوع FM با n عضو مستقل تنها درحالی‌که مقادیر بیت‌های اطراف $S(M)[\log_2 n]$ دارای واریانس زیادی باشد، به‌صورت پیشوندی از تمام یک و پسوندی از تمام صفر ظاهر می‌شود. با استفاده از این ویژگی طول S(M) به‌دست می‌آید. بدین ترتیب تنها با داشتن طول پیشوند همه یک‌ها، می‌توانیم n را تخمین بزنیم.

۵-۴- پردازش تحلیلی برخط بلادرنگ مقیاس‌پذیر روی معماری‌های ابری

برای این منظور یک ساختار اندیس‌گذاری به نام درخت PDCR معرفی کرده است. این درخت تعمیم‌یافته درخت PDC و درخت DC می‌باشند. درخت DC یک مکعب داده با ابعاد سلسله‌مراتبی (مانند کشور-شهر-مشتری) را برای یک سیستم تک پردازش‌گر اندیس‌گذاری می‌کند. در درخت PDC قابلیت پردازش موازی بر روی یک پردازنده چندهسته‌ای اضافه شده است. درخت PDCR یک محیط توزیع‌شدهٔ ابری از درخت PDC را پیاده‌سازی کرده است. این درخت m+1 پردازنده چندهسته‌ای دارد که هرکدام k ریسمان موازی را اجرا می‌کنند. یکی از m+1 پردازنده به‌عنوان کارفرما[6] و بقیه m پردازنده به‌عنوان کارگر[7] تعیین می‌شوند. در کارفرما تنها عمل پرس‌وجو و در کارگرها هر دو عمل درج و پرس‌وجو انجام می‌شود. با افزایش اندازه بانک داده، پردازنده‌های دیگری در محیط ابری به‌صورت پویا اختصاص داده شده و m افزایش یافته و ساختار درخت دوباره مرتب می‌شود. بدین ترتیب اطمینان حاصل می‌شود که هردوی حافظه و پردازش‌گر با افزایش مقیاس بانک داده در دسترس خواهند بود [28]. همچنین، پردازش بی‌درنگ داده‌ها عظیم جریانی در [29] مورد توجه قرار گرفته و طراحی سامانه‌ای برای این منظور با بهره‌گیری از الگوریتم زمان‌بندی بی‌درنگ چندپردازنده‌ای مبتنی بر رویکر ترکیبی خوشه‌بندی ارائه شده است.

۵-۵- پردازش موازی جریان داده‌ها

در سامانه مدیریت جریان داده، برای هر پرس‌وجوی ثبت‌شده، نقشه‌ی پرس‌وجوی آن به‌صورت یک DAG ایجاد می‌شود. در نقشه عمل‌گرهای آن پرس‌وجو، توالی آن‌ها و نیز صف‌هایی برای بافرکردن تاپل‌های پردازش‌شده هر عمل‌گر

[6] hat

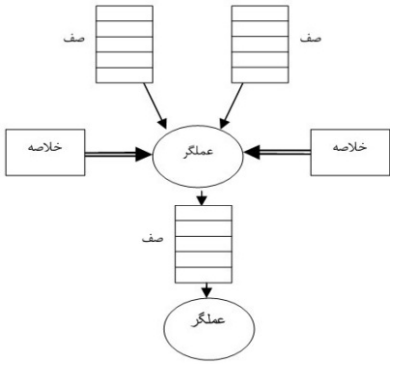
[7] worker

وجود دارد که این بافرها در عمل‌گر بعدی به‌عنوان ورودی مورداستفاده قرار می‌گیرند. (شکل-۹ (۹)) [30] و [31] .

با توجه به اینکه حجم داده‌هایی که قرار است، توسط هر پرس‌وجو پردازش شوند، بسیار زیاد است، بهبودی جزئی در نقشه پرس‌وجو و اجرای موازی آن، منجر به تأثیر چشم‌گیری در کارایی سیستم خواهد شد. برای پردازش موازی پرس‌وجو، پس از ایجاد نقشه پرس‌وجوی بهینه برای پرس‌وجوی ثبت‌شده، نحوه موازی‌سازی تعیین و درخت عملگرها که به‌طور موازی زمان‌بندی شده‌اند، برای اجرا روی داده‌ها به کار گرفته می‌شوند [32].

به‌طورکلی، برای موازی‌سازی، دو رویکرد زیر وجود دارد [32]:

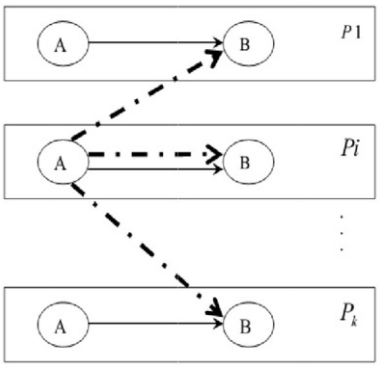
- **افراز داده‌ها**[1] : کل داده‌هایی را که باید پردازش شوند به تکه‌هایی افراز کرده و هر تکه را برای اجرا به یک سیستم تخصیص می‌دهیم. در این روش، همه ماشین‌ها کار یکسانی را روی داده‌های متمایز (افراز شده) انجام می‌دهند.
- **کنترل موازی**[2] : کل داده‌ها در اختیار هر یک از سامانه‌ها قرار دارند؛ اما هر سامانه بخشی از کار را روی داده‌ها انجام می‌دهند. برای اجرای موازی هر پرس‌وجوی ثبت‌شده روی k ماشین منطقی[3] ، ابتدا نقشه پرس‌وجوی آن ایجاد می‌شود؛ سپس k رونوشت به‌طوردقیق یکسانی از این نقشه ایجاد و برای هر ماشین منطقی ارسال می‌شود [33] و [34].



(شکل-۹): نقشه پرس‌وجو برای پرس‌وجوهای پیوسته جریان‌های داده

(Figure-9): Query plan for continuous queries on data streams

با فرض امکان ارتباط ماشین‌ها با یکدیگر، چنانچه در نقشه پرس‌وجو، عمل‌گر A باید نتایج پردازش خود را به عمل‌گر B بدهد، عمل‌گر A از ماشین i، تاپل‌های خروجی خود را هم می‌تواند به عمل‌گر B همان ماشین ارسال کند و هم به عملگر B در هر یک از ماشین‌های دیگر(شکل-۱۰ (۱۰)).



(شکل-۱۰): ارتباط بین ماشین‌های منطقی در اجرای عملگرهای نقشه پرس‌وجو

(Figure-10): The collaboration between logical machines in the execution of query plan operators

همان‌طور که مشخص است، این معماری از روش کنترل موازی جهت موازی‌سازی استفاده کرده است. همچنین در این مقاله روشی مبتنی بر مدل محاسباتی نگاشت-کاهش[4] (موازی‌سازی افراز داده‌ها) در کاربردهای داده - محور و پردازش دسته‌ای نیز ارائه شده است [35].

۶- نتیجه‌گیری و کارهای آینده

در این مقاله، برای پاسخ به پرس‌وجوهای پیوسته تجمعی اقتضایی راه‌حلی مبتنی بر درخت تجمعی پیشوندی ارائه شد که به‌صورت پویا ایجاد، نگهداری و مدیریت، و در هنگام پاسخ به پرس‌وجو مورد بهره‌برداری قرار می‌گیرد. هدف در طراحی و ارائه این راه‌حل آن بوده که به‌طور پویا، مؤثرترین نتایج تجمعی در ساختار درختواره ذخیره و مورد استفاده قرار گیرند تا موازنه[5] ای مناسب بین دقت نتایج از یک‌سو، و کارآمدی سیستم (از جهت برخط بودن و بهره‌وری در مصرف حافظه) از سوی دیگر تأمین شود.

همان‌طور که نشان داده شد، با استفاده از درخت پیشوندی پویا زمان اجرای الگوریتم (۲) (پاسخ به پرس‌وجو) به مرتب $O(n)$ ه نزدیک شده است و ضمن مصرف میزان قابل قبولی حافظه برای این کار، دقت مناسبی برای ارائه نتایج و پاسخ‌ها نیز تأمین خواهد شد.

[1] Data Partitioning یا Parallel Data

[2] Parallel Control

[3] Logical

[4] Map Reduce

[5] Trade-off

برای بهبود عملکرد درخت پویا لازم است با توجه به رفتار سیستم، نسبت به انتخاب مقدار مناسب برای φ اقدام کنیم. همان‌طور که درقبل گفته شد، برای سیستم‌هایی که تغییر رفتار زیادی دارند، روش درخت پویا، کارایی ندارد و باید بتوانیم با افزایش φ، خاصیت پویایی درخت را کم و بی‌اثر کنیم. انتخاب φ مناسب به‌عنوان یک متغیر تنظیم عملکرد درخت پویا می‌تواند در پژوهش‌های آتی مورد بررسی قرار گیرد.

تقدیر و تشکّر

از مدیریت شرکت جویا افزار ماندگار پرسیا که در زمان تهیه و تدوین این مقاله، حمایت‌ها و پشتیبانی های مالی و معنوی خود را دریغ ننمودند، کمال تشکر و قدردانی را دارم.

7-References ۷- مراجع

[1] M. Chen, S. Mao and Y. Liu, "Big Data: A Survey," *Mobile Networks and Applications,* vol. 19, no. 2, pp. 171-209, April 2014.

[2] C. L. Philip Chen and C. Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," *Elsevier,* vol. 275, pp. 314-347, January 2014.

[3] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein and R. Varma, "Query Processing, Resource Management, and Approximation in a Data Stream Management System," in *Proceedings of the 2003 CIDR Conference*, Asilomar, 2003.

[۴] صباغ‌گل ریحانه، دانشپور نگین. بهبود الگوریتم انتخاب دید در پایگاه داده تحلیلی با استفاده از یافتن پرس‌جوهای پرتکرار. پردازش علائم و داده‌ها. ۱۳۹۶؛ ۱۴ (۱): ۲۹-۴۰.

[4] R. Sabbagh Go and N. Daneshpour, "An Improved View Selection Algorithm in Data Warehouses by Finding Frequent Queries," *JSDP,* vol. 14, no. 1, pp. 29-40, 2017.

[5] M. Cho, J. Pei and K. Wang, "Answering ad hoc aggregate queries from data streams using prefix aggregate trees," *Knowledge and Information Systems,* vol. 12, no. 3, pp. 301 - 329, August 2007.

[6] C. J. Hahn, S. G. Warren and R. Eastman, "Extended Edited Synoptic Cloud Reports from Ships and Land Stations Over the Globe," 1952-2009. [Online]. Available: http://cdiac.ornl.gov/epubs/ndp/ndp026c/ndp026c.html.

[7] K. Beyer and R. Ramakrishnan, "Bottom-up computation of sparse and Iceberg CUBE," in *SIGMOD '99 Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, New York, 1999.

[8] K. A. Ross and D. Srivastava, "Fast Computation of Sparse Datacubes," in *VLDB '97 Proceedings of the 23rd International Conference on Very Large Data Bases*, San Francisco, 1997.

[9] Y. Sismanis, A. Deligiannakis, N. Roussopoulos and Y. Kotidis, "Dwarf: Shrinking the PetaCube," in *SIGMOD '02 Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, New York, 2002.

[10] W. Wang, J. Feng, H. Lu and J. Yu, "Condensed cube: an effective approach to reducing data cube size," in *18th International Conference on Data Engineering*, San Fransisco,CA, 2002.

[11] J. Li, D. Maier, K. Tufte, V. Papadimos and P. A. Tucker, "Semantics and evaluation techniques for window aggregates in data streams," in *International Conference on Management of Data*, New York, 2005.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong, "TAG: a Tiny AGgregation Service for Ad- Hoc Sensor Networks," *ACM SIGOPS Operating Systems Review - OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation,* vol. 36, no. SI, pp. 131-146, 2002.

[13] J. Considine, F. Li, G. Kollios and J. Byers, "Approximate aggregation techniques for sensor databases," in *ICDE '04 Proceedings of the 20th International Conference on Data Engineering*, Washington, DC, USA, 2004.

[14] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *MobiCom '00 Proceedings of the 6th annual international conference on Mobile computing and networking*, New York, NY, USA, 2000.

[15] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM SIGMOD Record,* vol. 31, no. 3, pp. 9-18, September 2002.

annual ACM symposium on Parallel algorithms and architectures, New York, NY, USA, 2001.

[28] F. Dehne, Q. Kong, A. Rau-Chaplin, H. Zaboli and R. Zhou, "Scalable real-time OLAP on cloud architectures," *Parallel and Distributed Computing,* Vols. 79-80, pp. 31-41, May 2015.

[29] A. A. Safaei, "Real-time Processing of Streaming Big Data," *Journal of Real-Time Systems,* August 2016.

[30] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom, "Models and issues in data stream systems," in *twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database*, New York, 2002.

[31] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, K. Ito, R. Motwani, U. Srivastava and J. Widom, "Stream: The stanford data stream management system," Stanford InfoLab, 2004.

[32] D. J. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Processing," *Communications of the ACM,* vol. 36, no. 6, pp. 85-98, June 1992.

[33] A. A. Safaei and M. Haghjoo, "Dispatching of Stream Operators in Parallel Execution of Continuous Queries," *Supercomputing,* vol. 61, pp. 619-641, 2012.

[34] A. A. Safaei and M. Haghjoo, "Parallel processing of continuous queries over data streams," *Distributed and Parallel Databases,* vol. 28, no. 2, pp. 93-118, 2010.

[۳۵]ع. ا. صفایی و م. حق جو, "پردازش موازی جریان داده ها," نشریه علمی پژوهشی انجمن کامپیوتر ایران, جلد ۱۱, شماره ۱ (الف), pp. 11-29, 1392.

[35] Safaei, A. A., and M. S. Haghjoo. "Parallel processing of data streams." J Comput Sci Eng 11.2 (2014): 11-29

علی اصغر صفائی متولد سال ۱۳۵۷، کارشناسی و کارشناسی ارشد خود را در رشته مهندسی کامپیوتر- گرایش نرم‌افزار به‌ترتیب در سال‌های ۱۳۸۰ و ۱۳۸۳ به پایان رسانید. او در سال ۱۳۹۰ موفق به دریافت درجه دکترا در همین رشته و در حوزهٔ پایگاه داده‌ها از دانشگاه علم و صنعت ایران شد. وی هم‌اکنون به‌عنوان عضو هیئت علمی گروه انفورماتیک پزشکی دانشگاه تربیت مدرس مشغول به فعالیت است. زمینه‌های پژوهشی ایشان شامل سامانه‌های

[16] Y. Yao and J. Gehrke, "Query Processing for Sensor Networks," Asilomar, 2003.

[17] J. Zhao, R. Govindan and D. Estrin, "Computing Aggregates for Monitoring Wireless Sensor Networks," in *SNPA*, 2003.

[18] S. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong, "The Design of an Acquisitional Query Processor for Sensor Networks," in *International Conference on Management of Data*, New York, NY, USA, 2003.

[19] A. A. Safaei, M. Haghjoo and F. Abdi, "Semantic Cache Schema for Query Processing in Mobile Databases," in *IEEE ICDIM'08*, London, UK, 2008.

[20] A. A. Safaei, K. Mehran, M. Haghjoo and K. Izadi, "Caching Intermediate Results for Multiple-Query Optimization," in *5th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-2007)*, Jordan, 2007.

[21] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences,* vol. 31, no. 2, pp. 182 - 209, 1985.

[22] N. Alon, Y. Matias and M. Szegedy, "The space complexity of approximating the frequency moments," *JOURNAL OF COMPUTER AND SYSTEM SCIENCES,* vol. 58, no. 1, pp. 137-147, February 1999.

[23] Z. Bar-yossef, T. S. Jayram, R. Kumar, D. Sivakumar and L. Trevisan, "Counting Distinct Elements in a Data Stream," in *Randomization and Approximation Techniques in Computer Science*, Springer-Verlag, 2002, pp. 1-10.

[24] G. Cormode, M. Datar, P. Indyk and S. Muthukrishnan, "Comparing data streams using hamming norms (how to zero in)," *IEEE Transactions on Knowledge and Data Engineering,* vol. 15, no. 3, pp. 529-540, March 2003.

[25] P. Flajolet, "On Adaptive Sampling," *Computing* , vol. 43 , no. 4, pp. 391 - 400, February 1990.

[26] S. Ganguly, M. Garofalakis and R. Rastogi, "Processing set expressions over continuous update streams," in *MOD International Conference on Management of Data*, New York, NY, USA, 2003.

[27] P. B. Gibbons and S. Tirthapura, "Estimating simple functions on the union of data streams," in *SPAA '01 Proceedings of the thirteenth*

پایگاه داده و جریان داده، داده‌های عظیم، پردازش موازی و بی‌درنگ پرس‌وجوها، حافظه‌های نهان‌معنایی، امنیت در سامانه‌های پایگاه داده و مدیریت اطلاعات در مقیاس وب است. از ایشان کتاب و مقالات متعدّدی نیز در همین زمینه‌ها به چاپ رسیده است.

نشانی رایانامه ایشان عبارت است از:

aa.safaei@modares.ac.ir

مهدی مسافری متولد سال ۱۳۵۷، کارشناسی و کارشناسی ارشد خود را در رشته مهندسی کامپیوتر-گرایش نرم‌افزار به‌ترتیب در سال‌های ۱۳۸۰ و ۱۳۸۹ به پایان رسانید. وی از ۱۳۹۰ با شرکت جویا افزار ماندگار پرسیا در تحلیل و طراحی نرم‌افزار برنامه‌ریزی منابع سازمان (ERP) همکاری داشته و هم‌اکنون مدیریت بخش بانک‌های اطلاعاتی این شرکت را عهده‌دار است. موضوعات مورد علاقه ایشان، بانک‌های اطلاعاتی، داده‌کاوی، پایگاه داده تحلیلی، جریان داده‌ها و مهندسی نرم‌افزار است.

نشانی رایانامه ایشان عبارت است از:

mehdimosaferi@gmail.com