WUJNS
Wuhan University Journal of Natural Sciences

Vol.11 No.1 2006 133-137

Article ID:1007-1202(2006)01-0133-05

A Granularity-Aware Parallel Aggregation Method for Data Streams

WANG Yong-li, XU Hong-bing†, XU Li-zhen, QIAN Jiang-bo, LIU Xue-jun

Department of Computer Science and Engineering, Southeast University, Nanjing 210096, Jiangsu, China

Abstract: This paper focuses on the parallel aggregation processing of data streams based on the shared-nothing architecture. A novel granularity-aware parallel aggregating model is proposed. It employs parallel sampling and linear regression to describe the characteristics of the data quantity in the query window in order to determine the partition granularity of tuples, and utilizes equal depth histogram to implement partitioning. This method can avoid data skew and reduce communication cost. The experiment results on both synthetic data and actual data prove that the proposed method is efficient, practical and suitable for time-varying data streams processing.

Key words: data streams; parallel processing; linear regression; aggregation; data skew

CLC number: TP 311

Received date: 2005-05-10

Foundation item: Supported by Foundation of High Technology Project of Jiangsu (BG2004034), Foundation of Graduate Creative Program of Jiangsu (xm04-36)

Biography: WANG Yong-li (1974-), male, Ph. D. candidate, research direction: data streams processing, knowledge discovery, hardware and software blending. E-mail: wyl_seu@126.com

† To whom correspondence should be addressed. E-mail: hbxu@seu.edu.cn

0 Introduction

Data streams in many applications, such as web service, traffic control, weather forecast etc, come from distributed data sources at different geographic locations. Distributed processing is the inevitable development trend for managing data streams. Shared-nothing clusters can scale up to thousands of computers, increase the available main memory, processors, disk space and bandwidth along the way, and thereby provide potential for high throughput and low latencies. Yet to date(As yet), the shared-nothing approach has been overlooked for continuous query (CQ) systems[1]. CQ systems challenge traditional parallelism techniques because of the delay of the communication; those tuples that have been partitioned out may cause inconsistency when expiration. They require adaptive, online repartitioning, and load balancing of lookup-based operators.

At present, there have been a lot of research work in parallel processing of time series, common data partitioning method is on the basis of overlap time-line strategy[2], typical time sequence data such as stock price usually keeps intact within one period, however, each attribute value of tuples in data streams is possible to be inconsistent with each time unit. We consider data streams are a general form of time series, so the partition strategy applies to time series is different with partition strategy for data streams. The fundamental difference of the data partitioning characteristics between data streams and static relation data sets is whether the tuples that have been partitioned can reappear. It is the decisive influence for data repartition tactics when data skew occurs and the attribute is involved in query changes.

We propose a parallel aggregation method for data steams

based on shared-nothing architecture. Because aggregation is the most important operation in many applications, we mainly focus on aggregation operation in this paper. We have realized all kinds of aggregation that are described in Ref. [3]. These kinds of aggregation are sufficient for more extensive and complicated operations.

1 Granularity-Aware Parallel Aggregation Model

1.1 Coordinator-Worker Architecture

As a realization of our proposed method, we construct a parallel aggregation model called coordinator-worker. Its architecture is composed of the coordinator (Data Engine) and the worker (Front-end) with the query preprocess layer in the hardware. It uses two-phase scheduling. Firstly, by analysing the approximate distribution of each data stream, we use variable partition granularity to reduce the cost of communication and distribute equal computing load to each processor. Secondly, after the query task and data streams have been distributed, we apply repartition correction factor to implement load shedding dynamically.

The architecture of worker-coordinator is illustrated in Fig. 1. Each site possesses its own local memory and disk, and communicates via messages with each other. The coordinator is responsible for receiving the query request from users, parsing the continuous query, generating the query plan and the data stream partitioning strategy. It performs distributed scheduling, load shedding, monitoring of the quality of the service and outputting the final query result to users. This central scheduling can

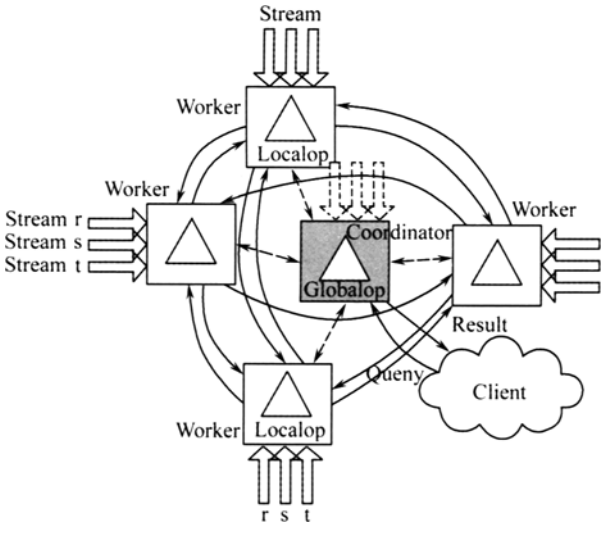

Fig. 1 Coordinator-worker architecture

reduce the frequency of communication among sites. The coordinator utilizes idle time to process its own task, store the information of the task and request from the worker into different queues. Each site can receive simultaneous data streams. It is redistributed to the other appropriate workers in all-to-all way in control of the coordinator. Each worker uses the same time window for the same query, and then executes the query operation involved in the content locally. The local query results are sent to the coordinator and are merged into the final result. Compared with the existing SCADA system whose front-end is only responsible for acquiring data rather processing complex tasks, this architecture distributes a lot of tasks to multiple intelligent front-end (worker) and intensively reduces the workload of the central server, so it can get better performance.

The key design issue includes data stream partition strategy, approximate algorithm, adaptively resource management, parallel continuous query optimising, etc. This paper focuses on data partition strategy appropriate for data streams. We list the key steps for the parallel aggregating algorithm based on the coordinator-worker model. This is a complete query plan:

① The same data stream is sampled in each site and samples are sent to the coordinator;

② The coordinator computes approximate variable partition granularity according to time-window in CQ and samples;

③ The coordinator computes partition vector utilizing attributes of samples involved in CQ. Then partition vector is sent to all worker;

④ Each worker redistributes stream tuples into the other worker on the basis of partition vector;

⑤ Each worker computes local aggregation in parallel over redistributed data streams;

⑥ Each local aggregation result is sent to the coordinator, which generates the global aggregation and outputs the final result. These steps proceed in cycles;

⑦ For limitation of space, we omit realization details of sampling and aggregating in this paper.

1.2 Data Partition Strategy Adapted to Stream Aggregating

It is unrealistic for tuples to serve as the partition granularity for data streams in Shared-nothing environment. The parallel processing will become meaningless because of the high communication cost. In order to share the overlap window among multiple queries over the same

streams, we should guarantee that the tuples partitioned to each site at one time fall in as many time-windows as possible. The worker need not upgrade the tuples buffer frequently. One naive method to calculate the size of the share time-window is to fetch the union of the time-windows at a certain moment. The number of tuples in the longest time-window is regarded as the partition granularity. However in many application scenarios, the maximal time-window size may be very long, which will cause congestion to occur, and the system cannot feedback to user's query request in time.

Our method is to set up the regressing statistic model for the size of the time-window according to the limited observation sample, and utilize the average number of the tuples in time-window that results from this statistic model as the partition granularity, for load shedding and sharing. It uses the maximal time-window length as the scale of maintaining intermediate tuples for local operation in the worker in order to assign the main memory adaptively. Once beyond the threshold of consumed memory, the part intermediate result should be discard or be stored into persistent stores. The purpose of sampling and regression analyzing in some workers is illustrated in Fig. 2.

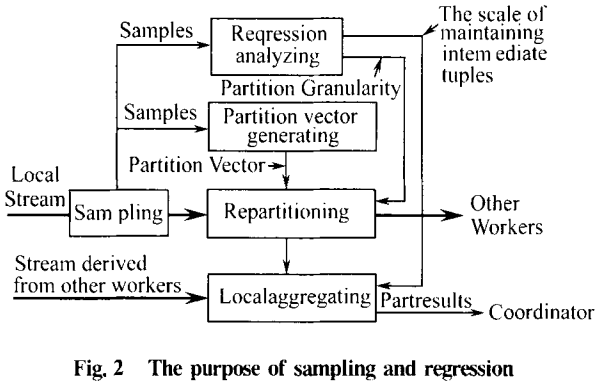

Fig. 2 The purpose of sampling and regression analysing in some worker

1.3 Stream Partition Relevant Definition

Consider a stream S containing an integer-valued attribute X. The value set V of X is the set of values of X that are present in S. For each $v \in V$, the frequency $f(v)$ is the number of tuples $t \in S$ with $t.X = v$. We assume that the elements of V have been sorted according to some sorting parameters, most commonly according to the numeric values of the v_i, i.e., $V = \{v_i \mid 1 \leqslant i \leqslant N\}$ where $i < j$ iff $v_i < v_j$. A histogram[4] of data distribution X is constructed by partitioning the frequency vector F of X into $B(\geqslant 1)$ intervals called buckets. In each bucket, we approximate the frequencies and values in some succinct fashion.

Definition 1 (Partition Vector $v[i]$). For $\forall i, 1 \leqslant i \leqslant k$, all records on site s_1 have key value less than $v[1]$, all records on site s_2 have key value greater than $v[1]$ but less than $v[2]$, and so on, until all records on site s_k have key value greater than $v[k-1]$. We say that $v[i]$ is the partition vector.

Definition 2 (Average Window Length $A(t)$ and Maximal Window Length $M(t)$). For any given timestamp t, we define that function $A(t)$ is the average length of time-window specified by user number at x, and $M(t)$ is the maximal length of time-window at t. and $M(t)$ is the maximal length of time-window at t. In order to estimate the value of $A(t_f)$ and $M(t_f)$ at future timestamp t_f, we observe n pair samples $(A(t_i), M(t_i))$ $i = 1, 2, \cdots, n$ during p time period from historical data steams, where t_i denotes timestamp, $A(t_i)$ denotes number of tuples in average length time-window, $M(t_i)$ denotes number of tuples in maximal length of time-windows. Suppose that the streaming velocity of tuples is relatively fixed, i.e., the value of $A(t_i)$ and $M(t_i)$ only depend on t_i. We first build regressing equation $E(y) = \alpha + \beta x$ (y denotes $A(t_i)$ or $M(t_i)$). Then we apply α' and β', the ordinary least square estimate of α and β, and experience regressing equation $y' = \alpha' + \beta' x$ to predict the value of y_f (y_f denotes $A(t_f)$ or $M(t_f)$) by equation $y_f = \alpha + \beta x_f + \varepsilon$. Where ε, called random error, is a random variable with 0 mean, which relates to abrupt events.

Definition 3 (Bin Slice (BS)). We say that partition granularity of tuples is Bin Slice, called BS, where $BS \sqsubset v[i]$ and $A(t) \rightarrow |BS|$, that is to say, the size of BS depends on the number of tuples in average time-window length.

Compared with fixed size partition granularity, variable partition granularity depending on $A(t)$ has better interactivity and flexibility. It can reflect the intent of user sufficiently.

Considering the expiration of data streams, the method based on range partition is more suitable for aggregate operation correlated with content, which can effectively avoid the data skew for data repartition. Our BSEDH(Bin Slice Equi-Depth Histogram) strategy is to sample over data streams; apply Equi-Depth Histogram technique to generate approximate partition vector over sample tuples; use the Bin Slice defined in definition 3 in every bucket as partition granularity; apply the method

of Round Robin Range Partition to redistribute tuples received locally to the other appropriator site.

This strategy is unsuitable for the situation where different aggregations are involved in different attributes of the same tuples. Because of the delay for redistribution may be greater than the expiration of the tuples. It will cause bad result and low efficiency. We can establish new partition vectors for new partition attributes, and duplicate the same tuples to the new site according to the tuples' attribute value range. In practical application systems, aggregation is generally involved in TupleID or Timestamp attribute. The attribute involved in aggregation in the same tuples seldom changes. The cost of replication incurred is acceptable. This tuples replication method can solve the above-mentioned problems.

2 Experiment Evaluation

In order to test the performance of the proposed model, we choose aggregate operation GROUP BY-SUM to test the above-mentioned method. A typical aggregation query as follows: **SELECT** AGG($t.X$) **FROM** S **WHERE** $t.X \in [6,12]$ **WINDOW** NOW- *w_length*. The experiments are conducted on an 8-node shared-nothing cluster of 2.66 GHz Pentium machines in which every node with 256 MB main memory and a 80 GB hard disk. Each machine was booted with version 2.4.18 of the Linux kernel. We used the LAM implementation of the MPI communication standard. With the LAM implementation, the average communication latency is 460 ms.

We use data generator in TelegraphCQ[5] software package to synthesize the uniform distributed data stream (Data set D_1) and use actual load data (Data set D_2) of power system from one area in Nanjing to test practical performance of this mode. We construct the discrete event generator to simulate Automation Generation Control system, which reads a tuple at variable intervals (for example millisecond) from above two data sets and sends it to our model. Suppose that the number of partition vector is set up to 8, which is equal to the number of sites.

Experiment 1 tests the relationship between the quantity of data streams and the parallel speedup. We test speedup of aggregating algorithm on tuple sets of different sizes (D_1), i.e. Corresponds to different streaming velocity. The experiment is run 10 times for four kinds of different data stream quantities, along with changing the number of sites from 2 to 8. Where $\varepsilon = 0.01$, we obtain the speedup vs. sites curve shown in Fig. 3. Labelled speedup is rate according to the running

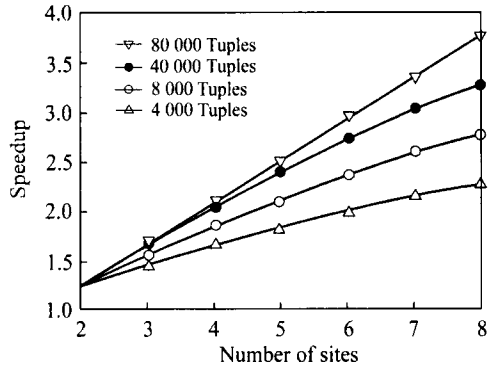

Fig. 3 Effect of data stream size on speedup (D_1)

time on each individual site. Fig. 3 indicates the heavier data stream size taken, the more the improved efficiency of aggregation algorithm using the partition method based on sampling and approximating. The intuitive reason for the good speedup is that adding more sites reduces the workload of aggregating on each site, but the workload of sampling on each site isn't changed.

For unlimited data streams, the time used by sampling only occupies a very small part of overall execution time on 8 sites. For example, compared with overall 220 seconds processing time for 80 000 tuples over data streams, sampling operation takes 2 seconds only.

Experiment 2 tests the aggregation throughput of this model over actual load data (D_2). Because the first partition granularity ——Bin Slice is related to submitted query only, we list the size of Bin Slice at the different time-stamp observed in Table 1.

Table 1 The size of Bin Slice at the different time-stamp.

t/s	5	10	15	20	25
Bin Slice/tuples	2 750	1 250	1 812	2 187	2 748

Experiment 3 examines how quickly the load-balancing policy reacts to a load perturbation by introducing an external load on a single machine. An extra process that computes for 0.15 s and sleeps for 0.1 s is used to exert the load. Fig. 4 illustrates the performance of our mini bin slice load shedding policy.

The top line in Fig. 4 shows the performance with no machines perturbed. The performance with load balancing turned off and load introduced at $t=1$ s. We report the aggregate throughput computed over one second interval. Suppose that the size of repartition correction factor mini Bin Slice is 500 (the number of streaming tu-

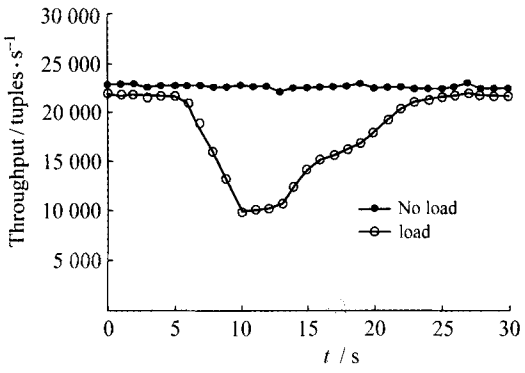

Fig. 4 Balancing processing load (D_2)

ples in a half of one second). Our load-balancing policy feels the effect of the imbalance caused by additional load after $t=5$ s as shown in the bottom curve. It begins to move min Bin Slice to react to this perturbation. It reaches steady state after offloading 30 min Bin Slices at $t=24$ s, it begins rebalancing after 14 s. The minimum time needed to transfer these partitions is 1 s. The dilation in reaction time is a result of two factors. One is the time to process and drain in-flight tuples before min Bin Slice movement occurs, and another is the collection phase, which waits as long as the previous move phase. Experiment 3 proves that this load balancing policy is efficient and feasible.

3 Conclusion

We propose a granularity-aware coordinator-worker model for parallel aggregating data streams. This model use a novel data partition method with the help of approximate techniques including sampling, linear regression and equal depth histogram etc, which supports variable repartition granularity for dynamic load balancing. The key idea of the granularity-aware method is to analyses the average (maximal) length of time-window specified by query and data distribution at every timestamp in a day based on history information. The experiments prove that the proposed model adapts to some application domains relatively fixed to the period of CQ, such as industry control or traffic control etc. Compared with the conventional control system this model can get higher performance and better flexibility by intelligent front-end (worker) with query processing layer. Improvement of data streams algorithm efficiency, and parallel query optimizing over data streams are on our future research agenda.

References

[1] Shah M, Hellerstein J, Chandrasekaran S, *et al*. Flux: An Adaptive Partitioning Operator for Continuous Query System. *Report No. UCB/CSD-2-1205*. Berkeley : University of California, 2002.

[2] Gray J, Bosworth A, Layman A, *et al*. Data cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub-Total. In: Su S Y W, Ed. *Proc of Intl. Conf on Data Engineering*, New Orleans: IEEE Computer Society, 1996. 152-159.

[3] Alin D, Minos G, Johannes G, *et al*. Processing complex aggregate queries over data streams. *Proc of the 2002 ACM SIGMOD Int'l Conf. on Management of Data*. New York: ACM Press, 2002. 61-72.

[4] Guha S, Koudas N, Shim K. Data-Streams and histograms. In: Yannakakis M, Ed. *Proc of the 33rd Annual ACM Symp on Theory of Computing*. Heraklion: ACM Press, 2001. 471-475.

[5] Chandrasekaran S, Cooper O, Deshpande A, *et al*. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. *Proc Conf on Innovative Data Syst Res*, Asilomar, CA, January 2003, 269-280.

□