► Routing and Maintenance
► Structure

Recommended Reading

1. Aberer K., Datta A., Hauswirth M., and Schmidt R. Indexing data-oriented overlay networks. In Int. Conf. on Very Large Data Bases (VLDB 2005). (More details at http://www.p-grid.org)
2. Aberer K., Datta A., and Hauswirth M. Multifaceted simultaneous load balancing in DHT-based P2P systems: a new game with old balls and bins. In Self-Properties in Complex Information Systems, Springer, Berlin, 2005.
3. Bharambe A., Agrawal M., and Seshan S. Mercury: supporting scalable multi-attribute range queries. In Symp. on Communications Architectures and Protocols (SIGCOMM 2004).
4. Brighten Godfrey P. and Stoica I. Heterogeneity and Load Balance in Distributed Hash Tables, Infocom 2005.
5. Byers J., Considine J., and Mitzenmacher M. Simple load balancing for distributed hash tables. In Second Int. Workshop on Peer-to-Peer Systems (IPTPS 2003).
6. Dabek F., Kaashoek F., Karger D., Morris R., and Stoica I. Wide-area cooperative storage with CFS. In ACM Symp. on Operating Systems Principles (SOSP 2001). (More details at http://pdos.csail.mit.edu/chord)
7. Datta A., Schmidt R., and Aberer K. Query-load balancing in structured overlays. IEEE Int. Symp. on Cluster Computing and the Grid (CCGrid 2007).
8. Ganesan P., Bawa M., and Garcia-Molina H. Online balancing of range-partitioned data with applications to peer-to-peer systems. In Int. Conf. on Very Large Data Bases (VLDB 2004).
9. Girdzijauskas S., Datta A., and Aberer K. Oscar: Small-world overlay for realistic key distributions. In the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2006).
10. Godfrey B., Lakshminarayanan K., Surana S., Karp R., and Stoica I. Load Balancing in Dynamic Structured P2P Systems, Infocom 2004.
11. Karger D., Lehman E., Leighton T., Levine M., Lewin D., and Panigrahy R. Consistent hashing and random trees: tools for relieving hot spots on the World Wide Web. ACM Symposium on Theory of Computing (STOC 1997).
12. Kleinberg J. The small-world phenomenon: an algorithmic perspective. In ACM Symp. on Theory of Computing (STOC 2000).
13. Mitzenmacher M. The power of two choices in randomized load balancing. IEEE Trans. Parall. Distrib. Syst., 12 (10):1094–1104, 2001.
14. Raab M. and Steger A. Balls into bins – a simple and tight analysis. In International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'98).
15. Steinmetz R. and Wehrle K. (eds.). Peer-to-Peer Systems and Applications. Springer Lecture Notes in Computer Science, vol. 3485, 2005. Chapters 9 & 10. (Book website http://www.peer-to-peer.info/)

Load Shedding

Nesime Tatbul
ETH Zurich, Zurich, Switzerland

Definition

Data stream management systems may be subject to higher input rates than they can immediately process with their available system resources (e.g., CPU, memory). When input rates exceed the resource capacity, the system becomes overloaded and the query answers are delayed. Load shedding is a technique to remove excess load from the system in order to keep query processing up with the input arrival rates. As a result of load shedding, the system delivers approximate query answers with reduced latency.

Historical Background

Load shedding is a term that originally comes from electric power management, where it refers to the process of intentionally cutting off the electric current on certain lines when the demand for electricity exceeds the available supply, in order to save the electric grid from collapsing. The same term has also been used in computer networking to refer to a certain form of congestion control approach, where a network router drops packets when its buffers fill up. More recently, load shedding has been proposed as a way to deal with overload in data stream processing systems [4].

Foundations

The goal of load shedding is to make sure that limited system resources operate below their capacity levels in case of unpredictable bursts in data arrival rates. This is achieved by selectively discarding some of the data items, thereby reducing the load at the expense of producing an approximate query answer. The main challenge in this problem is to minimize the loss in answer accuracy.

Assume a set of continuous queries Q, represented as a query plan of operators, where some of these operators may be shared among multiple queries. A set of inputs I feed these queries with streaming data, exerting a total load of $Load(Q(I))$ on a particular system resource with capacity C. A load shedding scheme must address the following key questions:

1. When to shed load? Conceptually, load needs to be shed whenever $Load(Q(I)) > C$.

2. Where to shed load? Load can be discarded at any point in the query plan. Dropping load at earlier points avoids wasting work; however, because of shared operators in the query plan, an early drop might adversely affect the accuracy of too many query answers.
3. How much load to shed? Just enough of the load at the chosen point(s) in the query plan must be shed so that the total resource demand gets below the available capacity with minimal total loss in accuracy.
4. Which data items must be discarded? The data items to be discarded should be chosen based on the approximation model and the properties of the operators in the query plan.

Furthermore, any load shedding scheme must have low run-time overhead in order not to further stress the limited system resources.

Various approaches have been proposed as solutions to the above listed issues. These approaches differ in their assumptions along several dimensions, including:

1. The limited resource under consideration (e.g., CPU, memory, communication bandwidth),
2. The way to reduce load (e.g., drop data, create summaries),
3. The approximation model/objective (e.g., maximum subset, minimum relative error, maximum throughput),
4. The query operator(s) under consideration (e.g., sliding window aggregates, windowed joins),
5. The data arrival model (e.g., stochastic models, temporal models),
6. The control loop (open vs. closed)
7. The system architecture (centralized vs. distributed).

Within the scope of the Aurora Project, Tatbul et al. have proposed a solution framework for load shedding which focuses on CPU as the main scarce resource, and discarding tuples by inserting special drop operators into the running query plan as the load reduction method [14]. The goal in this work is to minimize utility loss in query answers in terms of two alternative QoS (Quality of Service) dimensions: (i) percent tuple delivery, using a *random drop*, or (ii) output values delivered, using a *semantic drop*. A random drop discards tuples based on a drop probability, whereas a semantic drop does so based on a predicate on the tuple content. The earlier the load is reduced in a query plan, the larger is the saving in processing resources. However, shedding load early in a shared query plan may hurt the accuracy for multiple queries. To address this conflict, it is shown that load reduction should be applied either on the input streams, or on streams that immediately follow a shared operator in the query plan. Furthermore, these potential drop locations are ranked in terms of a metric, called *loss/gain ratio*. The drop location that causes the smallest QoS utility loss for the corresponding CPU processing power gained in return per unit drop of data, is preferred over the other drop locations with larger ratios. This way, the overall loss in QoS utility is minimized. For low run-time overhead, this work has proposed to pre-compute a set of load shedding plans based on system statistics, and insatiate these plans at run time based on the observed input rates. The proposed framework has also been extended to handle load shedding on windowed aggregation queries [15]. The key idea is to use a third type of drop operator, called a *window drop*, which semi-probabilistically discards load in units of windows instead of on a per-tuple basis. This way, window integrity can be preserved throughput a query plan, and query answers are guaranteed to be subsets of the original answers. An alternative to the window drop approach was earlier proposed by Babcock et al. [3]. This work also targets load shedding on aggregation queries under CPU constraints, but uses a different approximation model where the goal is to minimize the maximum relative error across all queries. Drops are applied on a per-tuple basis, leading to query answers with errors in their values. Window statistics and well-known statistical bounds such as the Hoeffding inequality are used to control these errors for a certain set of aggregate functions, including sum, count, and average. A close alternate to dropping tuples under CPU limitations is the selective processing approach proposed by Gedik et al. [8]. This work selectively processes tuples in the stream windows for join operators, in order to maximize the output rate or semantic utility of the query results, in the presence of variations in input rates as well as time correlations between two join inputs.

Load shedding can also be used to deal with memory limitations. Das et al. have focused on this problem for stream joins, where the maximum subset measure is used as the approximation metric [7]. This work assumes a frequency-based data arrival model and proposes two practical heuristics: (i) PROB, which drops tuples from an input stream which had the smallest frequency of occurrence on the opposite

stream in the past (assuming that those tuples are the least likely to produce join results also in the future); (ii) LIFE, which drops tuples from an input stream whose product of frequency of occurrence on the opposite stream and remaining window lifetime is the smallest (i.e., the goal is to avoid investing on soon to be expired tuples). Within the scope of the STREAM Project, Srivastava and Widom have proposed an alternative load shedding approach for windowed stream joins in memory-limited environments [12]. This work is based on an age-based data arrival model, where it is assumed that the rate at which a tuple produces join results is solely determined by its age, specified as an age curve. To deal with memory shortage, tuples of a certain age are selectively discarded from the join window to make room for others, which have higher expectation of producing matches. The goal here is again to maximize the size of the join result set. A secondary concern in this work is to be able to produce a random sample from the join in case that the join is followed by an aggregate. In this case, the final output will not be a subset of the exact answer, and the overall goal is then to minimize the relative error, in line with the work of Babcock et al. [3].

Jain et al. have proposed a load shedding approach to reduce the network bandwidth usage [9]. This approach is based on Kalman Filters which can be used to model data streams as processes with states that evolve over time. As new tuples arrive at a source site, it is checked if the current model installed at the remote server site can still answer the query within given precision bounds. If so, there is no need to send the new tuple to the server, i.e., it can be discarded. Otherwise, the server model has to be updated, hence the new tuple is transmitted. Adaptivity is achieved by adjusting model parameters to changing load characteristics.

Stream load can also be reduced based on creating summaries of data instead of discarding data. This idea was pursued by two different lines of work within the scope of the TelegraphCQ Project: (i) Reiss and Hellerstein have proposed a load shedding technique called *data triage*, where excess data is not dropped, but stored in synopsis data structures [11]. At the end of a well-defined query window, the stored synopses are processed through a shadow query plan to compute an approximate result on the summarized portion of the data. Finally, exact and approximate results are merged into one composite result for that query window. This works using an error model based on Minkowski distance. (ii) Chandrasekaran and Franklin have focused on hybrid queries that process live data streams in correlation with historical data archived on disk [5]. In this case, disk becomes the bottleneck resource. To keep processing of disk data up with processing of live data, disk data is organized into multiple resolutions of reduced summaries. Depending on the live data rates, the system picks the right resolution summary to use in query processing. This is a form of load shedding that tries to cut down from disk access cost using data summaries.

The NiagaraCQ Project has taken an integrated approach where load shedding is seen as an extension to continuous query optimization. Kang et al. use a unit-time-based cost model where total cost of join processing is broken into two components, one for each join direction [10]. The optimal index and join algorithm combination for each direction is determined so as to maximize query throughput. Under CPU and memory limitations, the optimizer determines the ideal rate for each input and accordingly places a random drop to control the input rates. Ayad and Naughton use a similar analytical cost model, but extend it to plans with multiple joins [2]. It is shown that if computational resources are enough, then all join plans have the same throughput, however, they may substantially differ in their resource utilization. If all of these plans are infeasible (i.e., lead to CPU overload), then load must be shed via random drops. The focus is on picking the right join plan, the locations on the plan to insert the drops, and the amount of drops. An interesting result shown in this work is that the optimal join plan (i.e., with the lowest utilization) when resources are sufficient is not necessarily the optimal plan (i.e., with the highest throughput) when resources are insufficient.

All of the above described approaches assume that stream processing is performed on a single server. The overload problem can also arise in distributed stream processing systems where queries are distributed onto multiple servers. In a distributed environment, there is load dependency among the nodes that are assigned to run pieces of the same query. As a result, shedding load at an upstream node affects the load levels at its downstream nodes, and the load shedding actions at all nodes along a query plan will collectively determine the quality degradation at the query end-points. Within the scope of the Borealis Project, Tatbul et al. have

modeled this problem as a linear optimization problem, and proposed two alternative solutions: (i) a centralized approach, where a coordinator node produces globally optimal plans with the help of an LP solver, and the rest of the nodes adopt their share of these global plans; (ii) a distributed approach, where nodes exchange metadata information (represented in the form of a feasible input table (FIT) which shows input rate combinations that are feasible for a given node) with their neighbors, and each node produces its own plan based on the available metadata [13]. Both of these solutions are based on the idea of pre-computing the load shedding plans in advance and storing them in a quadtree-based plan index. It is shown that the FIT-based plan generation is more efficient than its solver-based counterpart. Furthermore, the distributed solution is expected to be more responsive in dynamic environments due to its ability to incrementally update previously computed load shedding plans, reducing the amount of run-time communication needed among the nodes.

These approaches are all open-loop solutions in that the system load is periodically monitored and the load shedding algorithms are triggered as necessary. There has also been recent work that applies control-theoretic concepts to finer-grained adaptive load shedding on data streams [1,16]. These approaches are based on constructing a feedback loop that continually monitors the high-frequency variations in system parameters and makes the necessary adjustments in the load controllers accordingly. Such closed-loop approaches are shown to be more adaptive for input workloads with higher frequency fluctuations in stream data rates.

Load shedding finds use also in resource-intensive data stream mining applications. As argued by Chi et al. [6], in common data mining tasks such as classification and clustering of multiple data streams, the impact of load shedding on performance is not known a priori as the mining quality often depends on specific feature values observed in the stream in a non-monotonic way. This requires feature value prediction and adaptation. The Loadstar scheme uses a Markov model to predict the distribution of future feature values whose parameters are adaptively updated in time in order to maximize the classification quality under CPU constraints [6]. The high-level idea in this work is to allocate more resources to data streams that carry more uncertainty while shedding the ones whose class labels are more certain for the upcoming time window.

Key Applications

Load shedding can be used in all data-intensive streaming applications for which low latency answers can be more critical than full answer accuracy. These include sensor-based monitoring (e.g., habitat monitoring, bio-medical monitoring, weather monitoring, road traffic monitoring), RFID-based asset tracking, GPS-based location tracking, video-based security monitoring, and network traffic monitoring.

Future Directions

Load shedding in data stream management systems is currently an active area of research. A significant body of research results has been produced in this area since circa 2002. The future directions include development of new load shedding schemes for other sets of assumptions along the dimensions listed above. There is also a need to integrate the complementary and alternative solution schemes under a single framework, which could automatically select the right set of techniques for a broad range of system resources, based on the characteristics of the received workload as well as the application-specific quality of service criteria.

Cross-references

- ► Adaptive Stream Processing
- ► Approximate Query Processing
- ► Architectures and Prototypes
- ► Continuous Query
- ► Data Sampling
- ► Data Sketch/Synopsis
- ► Data Stream
- ► Data Quality Dimensions
- ► Data Quality Models
- ► Data Reduction
- ► Stream Mining
- ► Stream-Oriented Query Languages and Operators
- ► Stream Processing
- ► Stream Sampling
- ► Wavelets on Streams
- ► Window-Based Query Processing
- ► Windows

Recommended Reading

1. Amini L., Jain N., Sehgal A., Silber J., and Verscheure O. Adaptive Control of Extreme-scale Stream Processing Systems. In IEEE ICDCS Conference. Lisboa, Portugal, 2006.
2. Ayad A. and Naughton J.F. Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams. In ACM SIGMOD Conference. Paris, France, 2004.

3. Babcock B., Datar M., and Motwani R. Load Shedding for Aggregation Queries over Data Streams. In IEEE ICDE Conference. Boston, MA, 2004.
4. Carney D., Çetintemel U., Cherniack M., Convey C., Lee S., Seidman G., Stonebraker M., Tatbul N., and Zdonik S. Monitoring Streams - A New Class of Data Management Applications. In VLDB Conference. Hong Kong, China, 2002.
5. Chandrasekaran S. and Franklin M.J. Remembrance of Streams Past: Overload-Sensitive Management of Archived Streams. In VLDB Conference. Toronto, Canada, 2004.
6. Chi Y., Yu P.S., Wang H., and Muntz R.R. Loadstar: A Load Shedding Scheme for Classifying Data Streams. In SDM Conference. Newport Beach, CA, 2005.
7. Das A., Gehrke J., and Riedewald M. Approximate Join Processing Over Data Streams. In ACM SIGMOD Conference. San Diego, CA, 2003.
8. Gedik B., Wu K., Yu P.S., and Liu L. CPU Load Shedding for Binary Stream Joins. Springer Knowledge and Information Systems Journal, 2006.
9. Jain A., Chang E.Y., and Wang Y. Adaptive Stream Resource Management using Kalman Filters. In ACM SIGMOD Conference. Paris, France, 2004.
10. Kang J., Naughton J.F., and Viglas S. Evaluating Window Joins over Unbounded Streams. In IEEE ICDE Conference. Bangalore, India, 2003.
11. Reiss F. and Hellerstein J.M. Data Triage: An Adaptive Architecture for Load Shedding In TelegraphCQ. In IEEE ICDE Conference. Tokyo, Japan, 2005.
12. Srivastava U. and Widom J. Memory Limited Execution of Windowed Stream Joins. In VLDB Conference. Toronto, Canada, 2004.
13. Tatbul N., Çetintemel U., and Zdonik S. Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing. In VLDB Conference. Vienna, Austria, 2007.
14. Tatbul N., Çetintemel U., Zdonik S., Cherniack M., and Stonebraker M. Load Shedding in a Data Stream Manager. In VLDB Conference. Berlin, Germany, 2003.
15. Tatbul N. and Zdonik S. Window-aware Load Shedding for Aggregation Queries over Data Streams. In VLDB Conference. Seoul, Korea, 2006.
16. Tu Y., Liu S., Prabhakar S., and Yao B. Load Shedding in Stream Databases: A Control-Based Approach. In VLDB Conference. Seoul, Korea, 2006.

LOC METS

Prasenjit Mitra
The Pennsylvania State University, University Park, PA, USA

Synonyms

Metadata encoding and transmission standard; Library of congress METS

Definition

The Library of Congress (LOC) Metadata Encoding and Transmission Standard (METS) is an XML (Extensible Markup Language) based format used for encoding metadata. The metadata is used to markup digital library objects in a repository or for exchange across repositories. METS is a Digital Library Federation initiative that is a successor to the Making of America II project (MOA2).

Key Points

The MOA2 project attempted to provide encoding formats for descriptive, administrative, and structural metadata for text and image-based documents. (http://www.loc.gov/standards/mets/METSOverview.v2.html) The Digital Library Federation (DLF) sponsored the project and the National Endowment for the Humanities funded it. MOA2 involved discussions led by the University of California, Berkeley with participants from New York Public Library and the libraries of Cornell, Penn State, and Stanford universities. The project produced a Document Type Definition (DTD) that specifies a vocabulary and syntax for encoding digital objects. (http://www.lib.berkeley.edu/digicoll/bestpractices/mets_history.html) The library community realized that the MOA2 DTD was too restrictive, because, MOA2 did not provide some basic functionality required for multimedia objects like video and audio. METS arose from efforts to address these problems in MOA2.

Digital objects and the metadata needed to describe them are different from the metadata required for documents. Digital objects require structural metadata that indicates how the components of the object are glued together and technical metadata that specifies how the digital object was produced. For example, if a digital object contains image and text files, the structural metadata indicates the hierarchical structure of these objects and files. Furthermore, without these metadata, the authenticity of a digital object could be in doubt. METs allows specifications of the structural technical metadata, as well as metadata required for internal management and administration.

A METS document consists of seven sections:

- *METS header*: describes the document itself and publishes information about the creator, editor, etc.
- *Descriptive metadata*: both external, i.e., residing outside the document and internal.