TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 01/10 pp125-135
Volume 21, Number 2, April 2016

Wide Area Analytics for Geographically Distributed Datacenters

Siqi Ji and Baochun Li*

Abstract: Big data analytics, the process of organizing and analyzing data to get useful information, is one of the primary uses of cloud services today. Traditionally, collections of data are stored and processed in a single datacenter. As the volume of data grows at a tremendous rate, it is less efficient for only one datacenter to handle such large volumes of data from a performance point of view. Large cloud service providers are deploying datacenters geographically around the world for better performance and availability. A widely used approach for analytics of geo-distributed data is the centralized approach, which aggregates all the raw data from local datacenters to a central datacenter. However, it has been observed that this approach consumes a significant amount of bandwidth, leading to worse performance. A number of mechanisms have been proposed to achieve optimal performance when data analytics are performed over geo-distributed datacenters. In this paper, we present a survey on the representative mechanisms proposed in the literature for wide area analytics. We discuss basic ideas, present proposed architectures and mechanisms, and discuss several examples to illustrate existing work. We point out the limitations of these mechanisms, give comparisons, and conclude with our thoughts on future research directions.

Key words: big data; analytics; geo-distributed datacenters

1 Introduction

Processing large volumes of data, often called *big data analytics*, has been one of the most important tasks that most corporations need, established enterprises and start-up companies alike. As examples, corporations need to analyze logs from customer activities, make recommendations based on histories of user browsing or purchases, and deliver advertisements to those that may be most interested in them. In the era of big data analytics, the volume of data to be processed grows exponentially, and the need for processing such volumes of data becomes more pressing.

Modern datacenters are deployed around the world, in a geographically distributed fashion, to process large volumes of data in a distributed manner using data parallel frameworks, such as Apache Hadoop and Spark. Traditionally, these data parallel frameworks are designed to process data within the same datacenter, where jobs typically run within the same cluster, and the data to be processed is locally stored in the Hadoop Distributed File System (HDFS).

However, as the volume of data grows, storing such data within the same datacenter is no longer feasible, and they naturally need to be distributed across multiple datacenters. This is further motivated by the fact that the data to be processed, such as user activity logs, are generated in a geographically distributed fashion. It is more efficient to store the data where they are generated, perhaps in Apache Hive, a data warehouse infrastructure designed to query and analyze data in distributed storage. Since data to be processed are increasingly stored across multiple datacenters around the world, existing data parallel frameworks that are designed to work well in a local cluster, such as Apache

• Siqi Ji and Baochun Li are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto M5S 3G4, Canada. E-mail: {siqiji, bli}@ece.utoronto.ca.
∗ To whom correspondence should be addressed.
Manuscript received: 2016-03-03; accepted: 2016-03-10

Hadoop and Spark, no longer meet the pressing need for big data analytics across multiple datacenters. In the literature, the problem of processing data across multiple datacenters is often referred to as *wide-area data analytics*.

The naive solution to process data across multiple datacenters is to first migrate all the data to one datacenter, and then process them locally, as illustrated in Fig. 1. Naturally, the volume of data to be processed, in the order or terabytes, makes it costly and inefficient to perform such wide-area network transfers. First, such an approach consumes a significant amount of network bandwidth[1], which incurs a high monetary cost. Even if the corporation has no budgetary concerns, the capacity of the inter-datacenter wide-area network is not increasing at the same rate as the volume of data to be analyzed[2], and such a solution is not going to be sustainable over the long run. Finally, migrating all the data to one datacenter takes time, and the longer it takes, the worse the performance.

The problem of wide-area data analytics has been widely acknowledged in the recent literature, and a number of solutions have been proposed. In this paper, we will focus on several representative solutions in the literature towards this research direction. Due to the pressing need of processing large volumes of data across multiple geo-distributed datacenters, these proposed solutions are exciting and highly relevant, and may soon be utilized in real-world data analytic applications. We begin with a brief introduction of the background of batch and streaming processing frameworks. With several examples, we will then proceed to present the basic ideas at a high level of these proposed solutions, and compare them when the need arises. We will also analyze these solutions, and point out their limitations and disadvantages, and provide our insights towards future work.

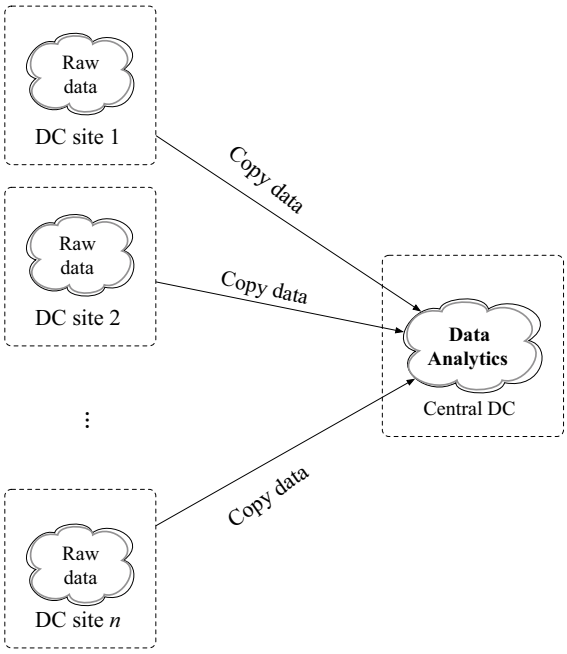


Fig. 1 Migrating all the data to one datacenter: A naive solution for wide-area data analytics across geo-distributed datacenters.

2 Background

In this section, we would like to briefly introduce the background of batch and stream processing frameworks.

2.1 Batch processing frameworks

When a short response time is not strictly required, *batch processing* is a widely used way to process considerable volumes of data without any user intervention. For batch processing, input data is collected beforehand, and then processed in batches.

Hadoop is a batch processing framework and data to be processed are stored in the HDFS[3], a powerful tool designed to manage large datasets with high fault-tolerance. *MapReduce*[4], the heart of Hadoop, is a programming model that allows processing a substantial amount of data in parallel. Figure 2 shows an example of the MapReduce model. It has three major processing phases: *Map*, *Shuffle*, and *Reduce*. Traditional relational database organizes data into rows and columns and stores the data in tables. MapReduce uses a different way, it uses key/value pairs. The

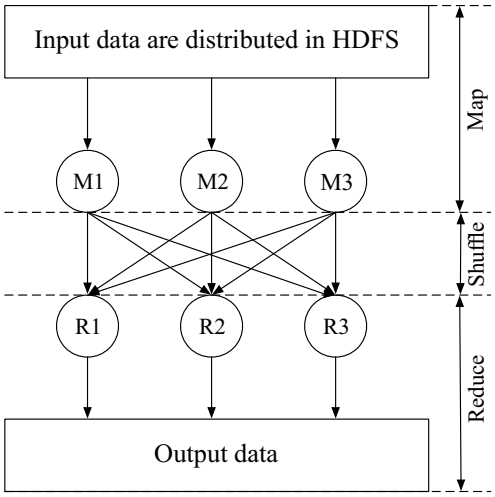


Fig. 2 Process of MapReduce: Map, Shuffle, and Reduce.

Map function performs sorting and filtering by keys, and then shuffles the intermediate results to the downstream operators which perform reduce tasks. The Reduce function applies summary operations on the intermediate data generated by Map.

Shuffle is one of the dependencies between the operators and their parents. Generally speaking, there are three kinds of dependencies: *One-to-one*, *Shuffle*, and *Join*[1]. One-to-one is the case when the node has only one parent and conversely the output of the node is consumed by at most one downstream operator. Shuffle has been shown in Fig. 2, which is the case when each node gives its output to all downstream nodes. When a node has either one-to-one dependency or shuffle dependency with each of its parent nodes, it is called Join.

Spark[5] has been a prevailing framework for batch processing since proposed in 2010. When it runs programs in memory, it achieves up to 100× faster than Hadoop. The upshot for Spark is it introduces an immutable, fault tolerant and parallel data structure called Resilient Distributed Dataset (RDD)[5].

The biggest problem for batch processing is high latency (for minutes), which is the delay between inputs and outputs. Furthermore, since batch processing deals with large volumes of data at one time, as long as there are any changes of the data, reprocessing of the batch job is required. Computations for batch processing could be complex due to the large data size.

2.2 Stream processing frameworks

The natural question that arises is: can we find a faster way to process data? *Stream processing* processes one data element or a small size of data in the stream at a time and the data are processed immediately upon arrival. For stream processing, computations are relatively simple and independent and it benefits from lower latency, typically seconds.

In order to support stream processing, *Spark Streaming* is proposed[6]. It divides the stream of data into batches of very small time intervals, which are defined as *Discretized Streams*. Spark Streaming is built on Spark and these Discretized Streams are treated as RDDs to perform computations. Strictly speaking, Spark Streaming can not do real stream processing but does micro-batching jobs. *Micro-Batching* is a special case of batch processing and it processes data with a very small batch size, which can be seen as a mix between batch processing and stream processing. Figure 3 shows the relations among batch processing, stream processing, and micro-batching. How to select a proper way to process data? It depends on the data size and requirements of the response time. Table 1 shows a comparison of these data processing approaches.

Apache Storm is a stream processing framework, it operates continuous stream of data. Apache Storm uses tuples, named lists of values, as its data model, and defines a stream as an unbounded sequence of tuples. Unlike Hadoop runs MapReduce jobs, Storm runs "topologies". A topology is a Directed Acyclic Graph (DAG) that users submit to Apache Storm for computations. Like Spark, Apache Storm is a fast, scalable, and high fault-tolerant parallel framework.

These frameworks are designed for the data processing within the same datacenter since they do not consider complicated situations of communication and scheduling in the wide-area data analytics. For example, in Spark, bandwidths across different sites are assumed to be uniformly distributed. Consequently, many representative works designed novel mechanisms for the wide area analytics based on these frameworks.

3 Optimization Issues

In order to achieve better performance, there are a lot

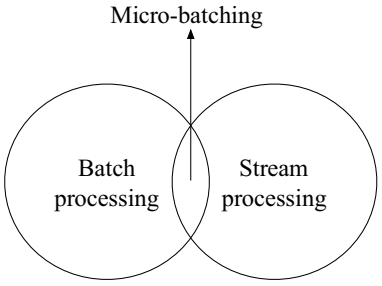


Fig. 3 Relations among batch processing, stream processing, and micro-batching.

Table 1 A comparison of data processing approaches.

	Data size	Latency	Computation	Examples
Batch processing	Large	High	Complex	Hadoop, Spark, Billing systems
Stream processing	Small	Low	Relatively simple	Apache Storm
Micro-batching	Small batch size	Low	Not so complex	Spark streaming

of optimization issues we need to consider in the wide-area data analytics.

Latency: In general, latency can be defined as a delay between receiving the request and generating the response. We build datacenters geographically with the purpose of achieving low latencies for local users[7]. Nevertheless, as data volumes keep increasing at a tremendous rate, it is still time consuming to transfer such substantial amount of data across datacenters[8]. Many cloud services have very stringent requirements for latency, even a delay of one second can make a great difference[9]. A large body of academic works have focused on optimizing latency.

Bandwidth: Bandwidth is the data transferred at one time. As bandwidth is scarce and expensive in the Wide Area Network (WAN)[10], optimizing bandwidth becomes another important issue in the analytics of geo-distributed data. Low latency may lead to the use of additional bandwidth, thus there is a tradeoff between bandwidth and latency. In this paper, bandwidth within the same datacenter is called intra-datacenter bandwidth, while bandwidth among different datacenters is called inter-datacenter bandwidth.

Fault-tolerance: High fault-tolerance is a big challenge when performing large scale data processing across datacenters. Fault-tolerance is the way that a system responses to a variety of network failures. A high fault-tolerant data processing system can continue operating when some components of the system fail[11], which can reduce costs and time for the reprocessing when failures happen.

Overhead: In order to achieve the optimal performance, sometimes we need to do extra work that can cause overhead. Overhead can be any excess resource like bandwidth, memory or computation time.

4 Mechanisms for Wide Area Analytics

Since the volume of data grows exponentially, the traditional centralized approach presents a number of limitations. In the wide area data analytics, data is generated in a geo-distributed fashion and some new constraints need to be considered, such as privacy concern.

Distributed execution is a strategy widely used in the wide area analytics. This strategy is to push computations down to local datacenters and then aggregate the intermediate results to do further processing. We use a motivation example to show the high-level idea of this strategy. A social network provider wants to get hot search words for every ten minutes. `Click_logs` and `search_logs` are two kinds of input data sources. `Click_logs` store web server logs of user activities and `search_logs` are the records of user requests for information. Base data is born distributed across datacenters, what we want to do is to give our execution strategy to minimize data traffic across different datacenters. If we use a centralized execution in Fig. 4, then we will observe that data traffic across datacenters is 600 GB per day. However, if we use a distributed execution strategy depicted in Fig. 5, then data size will be much smaller after preprocessing in the local datacenters, data traffic across datacenters is only 5 GB per day. Moreover, lower latency can be achieved by using the distributed execution.

Table 2 shows the summary of the wide area analytics. Many mechanisms are proposed for this problem. In this section, we will discuss high-level ideas of these representative mechanisms with some examples and give our thoughts about the proposed solutions.

4.1 Pixida

When a job is submitted for execution, we can get

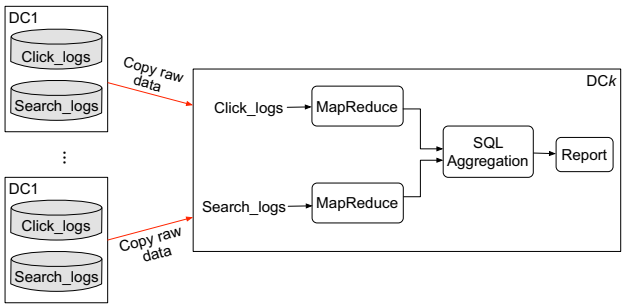


Fig. 4 A motivation example: Centralized approach.

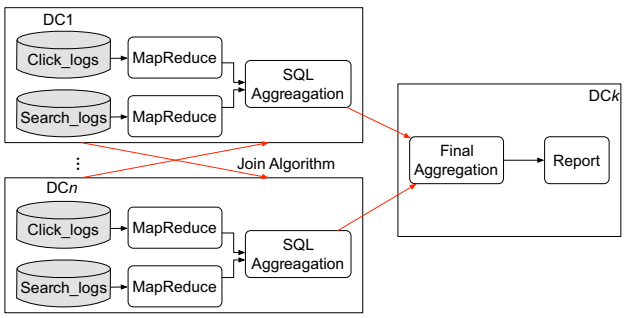


Fig. 5 Distributed execution of the motivation example: Preprocess in the local datacenters, then apply join algorithms to the intermediate results and do final aggregation to get the report.

Table 2 Summary of the wide area analytics.

Input data	Data is distributed across multiple datacenters.
Computational model	DAGs of tasks
Constraints	Privacy concern, data sovereignty and fault tolerance
Optimization issues	Latency, inter-datacenter bandwidth, overhead and cost

the job's task-level graph and locations of input data partitions from distributed storage systems like HDFS. Thus the data traffic minimization problem can be translated into the graph partitioning problem, where the job's task-level graph is split into partitions, each partition contains the tasks in the same datacenter. Intra-datacenter bandwidth is cheaper than inter-datacenter bandwidth, thus the objective is to minimize data traffic across different datacenters.

Pixida[1] is a scheduler designed to minimize data traffic across geo-distributed datacenters. This scheduler models the scheduling goal by using the graph partitioning method. It uses a new topology abstraction called "SILO", which is a group of nodes that belong to the same location. *Pixida* transforms the task-level graph into the SILO-level graph, which can reduce the size of the graph. Tasks for the same operator that run in the same location are merged into a single node. After getting the job's SILO-level graph, *Pixida* will assign tasks to different SILOs. Here, SILOs can also be regarded as datacenters.

It is obvious that the job's task-level graph and the locations of input data partitions can be known as soon as the job is submitted for execution, yet how to know the output data size of each task in the graph? *Pixida* designs the *Tracer* to solve this problem. In the Tracer phase, *Pixida* selects a sample of the input partitions like 20% to run the job, and then the Tracer extrapolates the output data size of each task.

Figure 6 gives an example of a job's SILO-level graph. In1, In2, and In3 represent input data partitions in three datacenters. Different graph partitions result in different inter-datacenter traffic. We use cost to represent the data transfer size across different datacenters. Traditionally, we could generate the graph partitioning problem as a min k-cut problem: "Given a weighted graph $G(V, E, W)$, and a set of k terminals, find a minimum weight set of edges E' such that removing E' from G separates all terminals"[1]. Here k represents k SILOs (datacenters). We can solve this

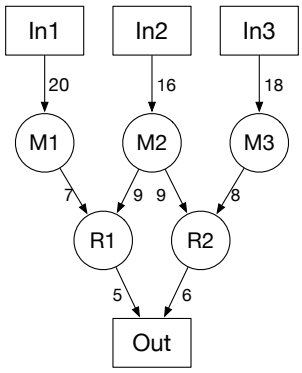


Fig. 6 A job's SILO-level graph, with the output data size of each task based on the statistics of the Tracer phase.

traditional min-k cut problem by using the Edmonds-Karp algorithm, and get the partitions with a cost of $7 + 8 + 5 + 6 = 26$ in the left side of Fig. 7.

However, we can make it "cheaper". Here we did not consider the case of "Dataflow Forking", when an operator forwards its output to more than one downstream operator. *Pixida* formulates a generalized min k-cut problem and presents a novel flow-based approximation algorithm (follows the structure of Edmonds-Karp algorithm) to solve this problem. The basic idea of solving the case of "Dataflow Forking" is to add an extra vertex V_e between M2 and its children in the graph. Figure 8 shows this idea. Edge (M2, V_e) represents a single cross-SILO transfer of the same data. Thus we could get the optimal graph partitioning in the right-side of Fig. 7, it has a cost of $7+9+8 = 24$, which is better than the left-side partitioning. As R1 and R2 are in the same SILO S4, one cross-SILO transfer from M2 in SILO S3 to R1 or R2 is enough.

Pixida is appropriate for batch processing. After integrated with Spark, it achieves up to 9× bandwidth

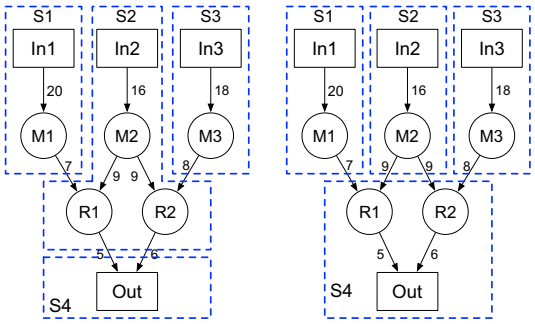


Fig. 7 Partitions of the job's SILO-level graph. The left-side partitioning has a cost of 26, which does not consider the case of "Dataflow Forking". The right-side partitioning is optimal with a cost of 24.

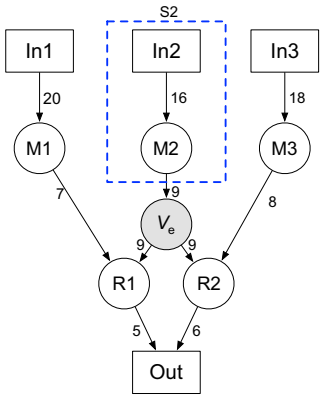


Fig. 8 An extra vertex is added between M2 and its children.

reduction compared with Spark. However, there are also some constraints about the graph partitioning method. First, it assumes that cross-SILO transfers are considered of equal cost. When we consider a complex pattern, the partitioning problem will become more complicated. Besides, it can not be used for real-time processing where data is processed upon arrival, since the input data partitions are static. Moreover, the Tracer phase used in *Pixida* adds computational and time overhead. Finally, *Pixida* only considers data transfers across datacenters, it does not tackle with the issue of latency.

4.2 WANalytics

WANalytics[12] is a Hadoop-based system, it also targets in minimizing inter-datacenter traffic. It is an extended version of *Pixida*.

Caching: *WANalytics* uses the idea of "caching" to cache all intermediate results to reduce data transfers. Figure 9 shows the basic idea of caching. At start, DC2 asks DC1 for the result of running query q_0, then DC1 runs the query and sends the result of q_0 to DC2. In the meantime, DC1 also caches the copy of q_0's result. If DC1 asks DC2 for the result of a new query q_1, then DC2 runs the query but only sends the difference of q_0's result and q_1's result. By this way, data transfers across datacenters are greatly decreased when there are repetitive queries.

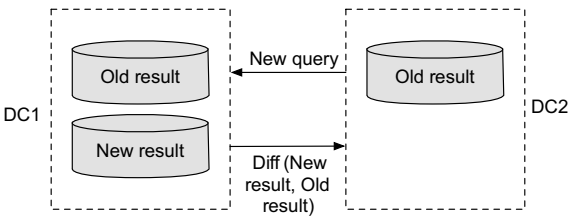


Fig. 9 Data transfer optimization: Caching.

This caching method actually worsens CPU and storage use, it solely reduces data transfers across different datacenters. Distributed queries are always seen as DAGs, when multiple DAGs share common subqueries, this method greatly helps to reduce data transfers as a result of sending the difference between new results and old results.

Optimizing execution: Given a set of recurrent DAGs of tasks, with constraints of sovereignty, *WANalytics* uses a greedy heuristic for the optimizing execution. It processes all DAGs in parallel. In each DAG, it goes over tasks in the topological order, and greedily chooses the lowest-cost available strategy for each task. There are two things the optimizing execution decides: (1) Strategy for each task, e.g., hash join or semi-join; (2) Which task goes to which datacenter, like the task graph partitioning problem.

Pseudo-distributed measurement: Similar as the Tracer phase in *Pixida*, Pseudo-distributed measurement is used to measure the cost of each execution strategy for a DAG. For some settings, measuring all options considered by the greedy heuristics could be very slow, which is a limitation for this measurement.

Figure 10 shows the architecture of *WANalytics*. *WANalytics* consists of two main components: a runtime layer and a workload analyzer. In the runtime layer, there is a coordinator in a master datacenter that interacts with datacenter managers at each datacenter. In each datacenter manager, there is a caching mechanism. Analyst submits DAGs of queries, and then the coordinator asks the workload analyzer for the best distributed execution plan.

After getting DAGs of queries, the workload analyzer

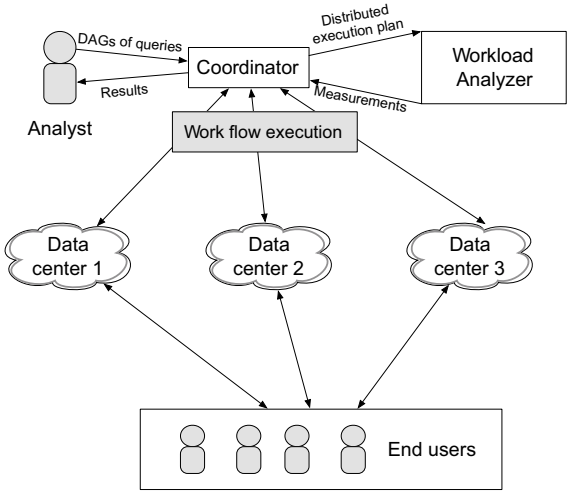


Fig. 10 *WANalytics*[12] architecture.

gives a distributed execution plan by using greedy heuristics and pseudo-distributed measurements. The workload analyzer uses a robust evolutionary approach. It starts by supporting the existing centralized approach, then uses a continuous adaptation: It firstly comes up with some DAG execution plans of the workload, secondly measures their costs by using pseudo-distributed measurements, then computes a new best plan by using the optimizing execution, finally it deploys the new best plan.

WANalytics focuses on the optimization of data transfers, but fault tolerance and latency are not addressed. Besides, this system only partially supports the requirement of data sovereignty. It considers data storage requirements but allows arbitrarily queries on the data.

4.3 Geode

Geode is a system built upon Hive that presented in Ref. [2]. It is an extended version of *WANalytics*[12]. This system uses a relational model and supports SQL analytics on the geo-distributed data. Figure 11 shows its architecture, which is similar as *WANalytics*. The core of *Geode* is the command layer. The command layer receives SQL queries, then gets a distributed query execution plan from the workload optimizer. After getting the plan, the command layer runs the plan and outputs results.

In the workload optimizer, *Geode* gives the query execution plan by solving an Integer Linear Program (ILP). The objective function of the ILP is to minimize total data transfers in the DAG. Constraints are the requirements of fault tolerance and data sovereignty. *WANalytics* uses the greedy heuristic for determining the query execution plan, but in some cases the heuristic approach fails to get the optimal solution. However, the ILP can only support up to ten datacenters in the experiment, greedy heuristic scales much better than the ILP. Moreover, the ILP is more accurate but it runs slower than the greedy heuristic. *Geode* gives a tradeoff between the running time and the solution quality.

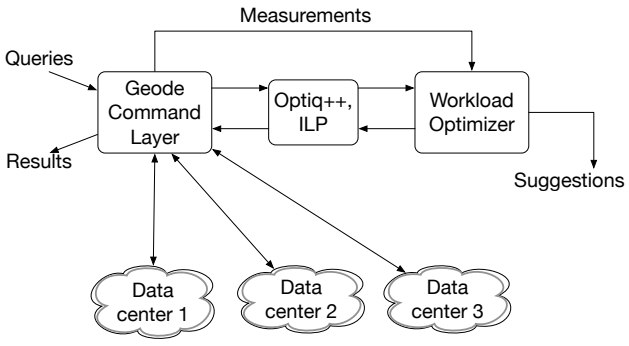


Fig. 11 ***Geode***[2] **architecture.**

4.4 Iridium

Pixida, *WANalytics*, and *Geode* only consider minimizing data transfers across datacenters but ignore the issue of latency, which is significant for low-latency processing applications. *Iridium*[13] is a system for the low latency geo-distributed analytics. It uses the task placement to reduce query response time. Figure 12 shows an example of simple MapReduce tasks across datacenters. DC1 and DC2 are two datacenters. DC1 has a downlink bandwidth of 100 MB/s, but its uplink bandwidth is low and only 10 MB/s. DC1 downloads the data of DC2 to do the Map task, and then generates a substantial amount of the intermediate data. There are no reduce tasks in DC1, so it needs to upload the intermediate data to DC2 to do reduce tasks. Because of the very low uplink bandwidth, query response time will be affected significantly. We call the link with low bandwidth as the bottleneck link. Usually, query response time is determined by the bottleneck link. If we could put more reduce tasks in DC1, there will be less data uploaded to DC2 via uplink, thus query response time will be greatly reduced.

This task placement problem can be formulated as a Linear Program (LP) with the objective of minimizing query response time. But the LP is appropriate for tasks of a single query. For DAGs of tasks, *Iridium* uses a greedy approach to perform the task placement by using the LP independently in each stage of the query. However, the best task placement is still limited by data locations. In order to better reduce query response time, *Iridium* also uses the data placement.

For the example in Fig. 12, we have already found the bottleneck link is the downlink of DC1, then

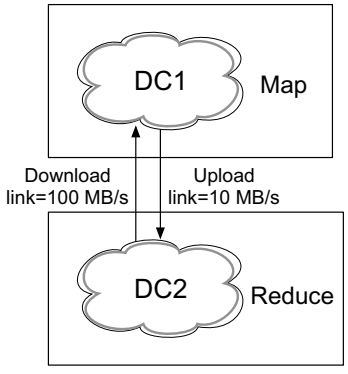


Fig. 12 An example of simple MapReduce tasks across datacenters.

if we could move the data out of DC1 to DC2 before the query arrives, query response time will be greatly reduced. For example, assume the next query will arrive in 24 s, the intermediate data in DC1 is 150 MB. Before the query arrives, we move all the intermediate data out of DC1 to DC2. Moving time is $150/10 = 15$ s, which is smaller than the query arrival time. For the data placement, the basic intuition is to identify the bottleneck link first, then move data out of the bottleneck site. When the intermediate data was generated at time t_0, and the next query arrives at time t_1, $t_1 - t_0$ is called the query lag. *Iridium* uses a greedy algorithm in the data placement, it seeks to move the high-value datasets and move the dataset that has the smallest query lag. For complex DAGs of tasks, sometimes results are not global optimal but local optimal. This data placement can be combined with the task placement, after completing the data placement before the query arrives, we could continue to use the task placement to reduce query response time.

One problem about data placement is how to estimate query arrivals? For repetitive queries, it is easy to estimate the query lag based on the old query. However, for other situations, it is hard to estimate. *Iridium* makes the following simple assumption that works well: for instance, if the dataset was generated at time t and two queries arrived at time $(t + a)$ and time $(t + b)$, and $a \leqslant b$. Then we will assume that the next two queries will arrive at time $(t + b) + a$ and time $(t + b) + b$.

One of the advantages of *Iridium* is that it supports both stream processing as well as batch processing. This system achieves low query response time by optimizing data and task placement, and it also considers the tradeoff between the bandwidth cost and query response time.

Task and data placement are also used in other academic works. NetStitcher[14] is a system for online data processing. It uses data placement to stitch unutilized bandwidth, and rescues up to five times additional bandwidth. Gu et al.[15] minimized the cost of servers and communication in geo-distributed datacenters, and formulated the cost minimization problem as a mixed-integer nonlinear programming to answer how to apply data and task placement under constraints of the remote data loading and quality of the service satisfaction.

4.5 SWAG

Finishing some tasks of a job does not mean faster job completion time. Job scheduling is another aspect that we need to consider in the wide area analytics. Hung et al.[16] targeted on reducing the average job completion time by using novel job scheduling algorithms, and achieved up to 50% improvement in the average job completion time with low overhead. They used two scheduling algorithms: Reordering and Workload-Aware Greedy Scheduling (SWAG).

Now we use an example to show the idea of these two scheduling algorithms. There are three jobs computed in three datacenters, Table 3 presents the sub-job sizes in each datacenter, which is the number of tasks for jobs. There are two assumptions in Hung et al.[16]: (1) Each datacenter has one computation source; (2) Each datacenter serves one task per second. Figure 13 shows the approach of First Come First Scheduling (FCFS) approach across three datacenters. Job order of FCFS is A $\rightarrow$ B $\rightarrow$ C, x axis represents the queue length (number of tasks to be served). If we use this FCFS scheduling, the average job completion time will be 13.3 s. However, we find that tasks of job A at DC2 has a higher completion time, then we may delay job A in favor of other jobs with faster completion time at dataceters. This is the basic idea of Reordering. Figure 14 shows this reordering approach, which moves some jobs later in the local queue, as long as delaying them will not increase the average job completion time. Reordering approach for our example achieves average job completion time of 13 s, which

Table 3 Sub-job sizes in datacenters.

Job arrival	DC1	DC2	DC3
A	2	10	3
B	8	4	1
C	6	3	7

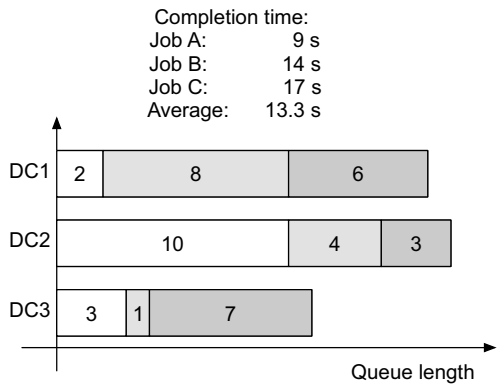


Fig. 13 First Come First Serve Scheduling, with the job order: A $\rightarrow$ B $\rightarrow$ C.

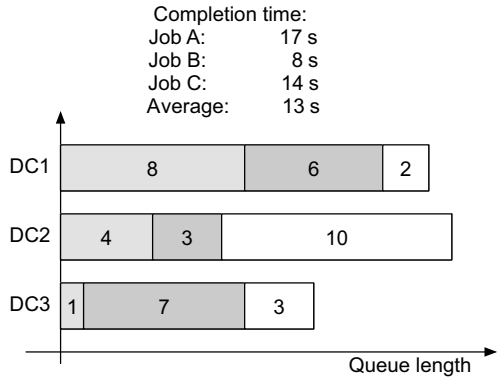


Fig. 14 Reordering Approach, with the job order: B → C → A.

is better than FCFS scheduling. The core idea of reordering approach is "no harm", which means that this approach provides "non-decreasing performance improvement for any scheduling algorithm[16]".

In order to achieve better average job completion time, Hung et al.[16] presented the SWAG algorithm. The basic idea for this algorithm is to greedily serve the job that finishes the fastest. When we schedule jobs by their finishing time, we also need to take the local queue size into consideration. For our example, job C finishes the fastest, so we serve job C first, and then the job B. Job A finishes the slowest, since it has 10 tasks to be finished in DC2, thus we serve job A at last. This scheduling approach by using SWAG achieves better average job completion time of 12.7 s than reordering. Figure 15 shows the idea of this scheduling method.

5 Discussion of Existing Mechanisms

Analytics for geo-distributed datacenters in the wide area network have several aspects. Some mechanisms are batch processing, some are stream processing. Bandwidth and latency are two important optimization issues we consider in the wide area analytics. Table 4 shows a comparison of these mechanisms we have discussed.

- *Graph Partitioning by Pixida*: This graph partitioning method is appropriate for the

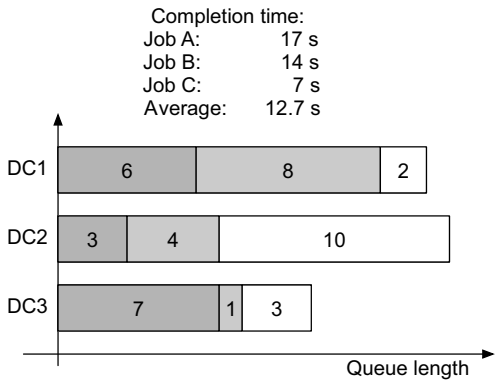


Fig. 15 SWAG algorithm, with the job order: C → B → A.

Table 4 A comparison of representative mechanisms.

Mechanism	Approach	Bandwidth	Latency	Overhead	Fault-tolerance	Limitations
Pixida	Graph partitioning	Optimizes inter-datacenter bandwidth	Does not consider	Does not consider	Does not consider	For simple communication patterns
WANalytics	Caching; Greedy heuristic for optimizing distributed execution	Optimizes inter-datacenter bandwidth but add bandwidth within one datacenter	Does not consider	Causes overhead for caching and computations within a datacenter	Does not consider	Data movement constraints are not considered; sometimes it is slow.
Geode	Caching; Integer Linear Program for optimizing distributed execution	Optimizes inter-datacenter bandwidth but add bandwidth within one datacenter	Does not consider	Causes overhead for caching and computations within a datacenter	Considered as a constraint	Data movement constraints are not considered; sometimes it is slow.
Iridium	Task and data placement	Considers the tradeoff between bandwidth and latency	Optimizes query response time	Low overhead	Does not consider	The greedy approach is not optimal for the general DAGs.
SWAG	Greedy job scheduling algorithm	Does not consider	Optimizes the average job completion time	Low overhead	Does not consider	The assumptions hide the complicated situations in the real-world data analytics.

simple task graph since cross-SILO transfers are considered to be equal, which does not cover the general cases in real life.

- *Distributed Query Planning*: *WANalytics* and *Geode* use a workload optimizer to find the best distributed execution plan. However, sometimes it may be slow for the workload optimizer to give the best execution strategy. Moreover, arbitrary queries are allowed on the data, which does not consider data movement constraints.
- *Task and Data Placement*: *Iridium* first finds the bottleneck link and then uses task and data placement to optimize query response time. However, it is hard to estimate the query lag, and sometimes the estimation will not be so accurate. Another limitation for the data placement is the data movement constraint, for some situations we can not move data out of a datacenter arbitrarily.
- *Job scheduling*: The idea is to schedule job by finishing time with the consideration of tasks at datacenters. It is simple but useful for optimizing the average job completion time. SWAG uses a greedy scheduling algorithm, yet it is not appropriate for a general job whose DAG consists of multiple stages. Furthermore, the assumptions for SWAG hide the complexity of real world data analytic jobs.

6 Conclusion

As data grows at a tremendous rate, achieving optimal performance in the wide area analytics becomes more and more challengeable. Compared with the local network in a datacenter, the WAN covers a relatively broad geographical area, which is more complicated and unstable. Moreover, processing a substantial amount of data within a very small time interval is a great challenge for those low latency cloud applications. In this paper, we present a number of typical mechanisms in the wide area analytics, discuss high-level ideas, and give a comparison of these mechanisms. Although with some limitations, more effective solutions may be inspired by these mechanisms and applied in the real world in the near future.

References

[1] K. Kloudas, M. Mamede, N. Preguiça, and R. Rodrigues, Pixida: Optimizing data parallel jobs in bandwidth-skewed environments, *VLDB Endowment*, vol. 9, no. 2, pp. 72–83, 2015.

[2] A. Vulimiri, C. Curino, P. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, Global analytics in the face of bandwidth and regulatory constraints, in *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, The hadoop distributed file system, in *Proc. of IEEE on Mass Storage Systems and Technologies (MSST)*, 2010.

[4] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in *Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[6] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, Discretized streams: Fault-tolerant streaming computation at scale, in *Proc. of the 24th ACM Symposium on Operating Systems Principles (SOSP)*, 2013, pp. 423–438.

[7] R. Couto, S. Secci, M. Campista, and L. Costa, Latency versus survivability in geo-distributed data center design, in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, 2014, pp. 1102–1107.

[8] Q. Zhang, L. Liu, K. Lee, Y. Zhou, A. Singh, N. Mandagere, S. Gopisetty, and G. Alatorre, Improving hadoop service provisioning in a geographically distributed cloud, in *Proc. of the 7th IEEE International Conference on Cloud Computing*, 2014.

[9] A. Munir, I. A. Qazi, and B. Qaisar, On achieving low latency in data centers, in *Proc. of IEEE International Conference on Communications (ICC)*, 2013, pp. 3721–3725.

[10] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, Aggregation and degradation in JetStream: Streaming analytics in the wide area, in *Proc. of USENIX NSDI*, 2014.

[11] P. Upadhyaya, Y. Kwon, and M. Balazinska, A latency and fault-tolerance optimizer for online parallel query plans, in *Proc. of ACM SIGMOD International Conference on Management of Data*, 2011, pp. 241–252.

[12] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese, WANalytics: Analytics for a geo-distributed data-intensive world, in *Proc. of Conference on Innovative Data Systems Research (CIDR)*, 2015.

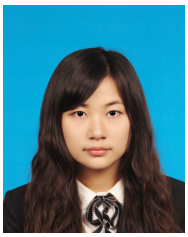
[13] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, Low latency geo-distributed data analytics, in *Proc. of ACM SIGCOMM*, 2015.

[14] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, Inter-datacenter bulk transfers with netstitcher, in *Proc. of ACM SIGCOMM*, 2011.

[15] L. Gu, D. Zeng, P. Li, and S. Guo, Cost minimization for big data processing in geo-distributed data centers, *IEEE Trans. on Emerging Topics in Computing*, vol. 2, no. 3, pp. 314–323, 2014.

[16] C.-C. Hung, L. Golubchik, and M. Yu, Scheduling jobs across geo-distributed datacenters, in *Proc. of the 6th ACM Symposium on Cloud Computing (SoCC)*, 2015.

Baochun Li received the BEng degree from Tsinghua University, China, in 1995 and the MS and PhD degrees from University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000, respectively. Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. Dr. Li has co-authored more than 280 research papers, with a total of over 13 000 citations. He was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a member of ACM and a Fellow of IEEE.



Siqi Ji received the BEng degree from Tsinghua University, China, in 2015. She is currently a first-year M.A.Sc. student in the iQua research group at the Department of Electrical and Computer Engineering, University of Toronto. Her current research interests include cloud computing and datacenter networking.