Journal of Big Data

SURVEY PAPER Open Access

A survey on bandwidth-aware geo-distributed frameworks for big-data analytics

Mohammed Bergui[1*], Said Najah[1] and Nikola S. Nikolov[2]

*Correspondence: mohammed.bergui@usmba.ac.ma
[1] Laboratory of Intelligent Systems and Applications, Department of Computer Science, Faculty of Sciences and Technologies, University of Sidi Mohammed Ben Abdellah, Fez, Morocco
Full list of author information is available at the end of the article

Abstract

In the era of global-scale services, organisations produce huge volumes of data, often distributed across multiple data centres, separated by vast geographical distances. While cluster computing applications, such as MapReduce and Spark, have been widely deployed in data centres to support commercial applications and scientific research, they are not designed for running jobs across geo-distributed data centres. The necessity to utilise such infrastructure introduces new challenges in the data analytics process due to bandwidth limitations of the inter-data-centre communication. In this article, we discuss challenges and survey the latest geo-distributed big-data analytics frameworks and schedulers (based on MapReduce and Spark) with WAN-bandwidth awareness.

Keywords: Hadoop, MapReduce, Geographically distributed clusters, Spark, WAN bandwidth, Cloud computing

Introduction

Nowadays, cloud service providers such as Microsoft, Amazon and Google deploy several data centres (DCs) across many locations in order to provide customers with fast access to their services. At these geographically distributed sites data is generated at high speed by applications such as sensor networks [1], climate science [1, 2], stock exchange [1], social networking applications [3, 4], log files from distributed servers [5], video stream from distributed cameras and scientific applications [1–3, 6]. Analysing massive amount of geo-distributed data is a daily requirement for decision-making by data analysts and real-time applications.

Big data processing frameworks such as MapReduce [7], Hadoop [8], Spark [9] and Dryad [10] have been designed to efficiently analyse large datasets. The problem is that all these frameworks assume single data centre deployment, where the network is generally available and homogenous. A trivial solution is to centrally aggregate all the data at one site, however analysing it may not be applicable due to privacy and regulatory constraints and has been proven a costly and wasteful use of WAN bandwidth [11, 12].

In a geo-distributed scenario, data is generated at different locations across the world and stored in multiple data centres. A geo-distributed data processing

framework should assign computation where data is located and then aggregate the outputs of these computations at a single site to execute the final computation as shown in Fig. 1.

Briefly, a Geographically Distributed big data Analytics (GDA) system should (1) execute jobs across different locations like a local job, transparent to the user, (2) support existing big data processing frameworks and languages, (3) allow movement of only the data relevant to the final output, (4) handle task, job, node, rack, and DC failure/outrage.

Recent efforts proposed new frameworks and scheduling techniques based on Hadoop, MapReduce and Spark that enable data analytics across multiple DCs [1, 6, 11, 13–16]. However, these frameworks are not optimised for the inter-DC bandwidth heterogeneity and limitations [17, 18]. In addition, most works assume that the sites have homogenous and available computational capacities which does not conform to the reality [19]. In this paper, we review the most popular and well-known frameworks that take into account WAN bandwidth in their problem formulation, since the cost and the performance of a geo-distributed job is dependent on WAN bandwidth and the amount of inter-DC data movement [20]. We believe that our survey can help both with the choice of a geo-distributed data analytics solution as well as with identifying the current open problems in engineering efficient bandwidth-aware geo-distributed analytics frameworks.

In this survey we only consider papers with the following criteria:

- Papers should include inter-DC bandwidth heterogeneity while improving application make span, query execution time or minimising inter-DC data transfers.
- Papers that only focus on geo-distributed scheduling or resource allocation mechanisms are excluded, eg. Flutter [16], WANanalytics [21], Pixida [22], Awan [23].

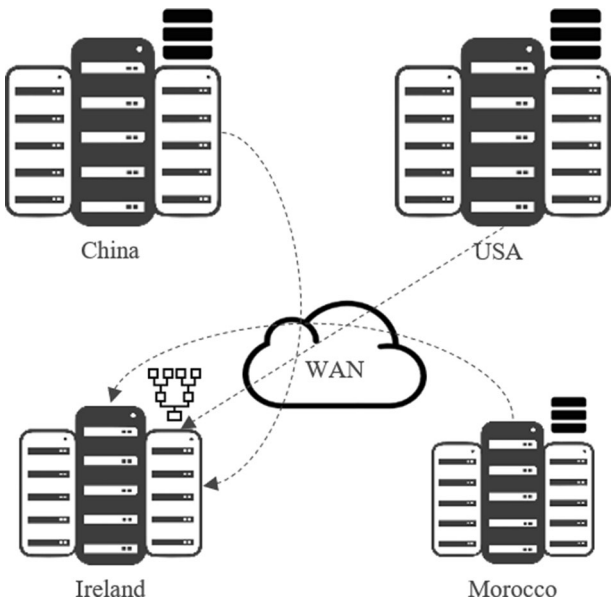


Fig. 1 A global overview of a geo-distributed big data processing

- Papers that focus on a particular type of data only are excluded, e.g. HPS+ [24].
- Papers that propose a GDA system that is based on a service available only by a specific cloud provider are excluded, e.g. Yugong [25].
- Frameworks that distribute data across different DC's before computation are excluded. e.g. Resilin [26], Photon [27]. The data should already be distributed before the computation.
- Geo-Distributed machine learning papers are excluded, e.g. Gaia [28].
- Papers published before 2014 are excluded.

Previous surveys in this domain include Dolev et al. [29] and Ji et al. [30], in this paper we both extend the scope of previous surveys with recent advances in geo-distributed computing and focus particularly on geo-distributed WAN-bandwidth aware big data frameworks. This reflects on the specific point of view we take on the motivations (see "Motivation" section) and challenges (see "Challenges" section) of geo-distributed computing as well as on the features for geo-distributed system categorisation (see "Geo-distributed big data processing" section).

The main goal of this paper is to provide organisations and researchers with a comprehensive review of geo-distributed big data processing systems that are efficient and could be deployed in production. Thus, we survey geo-distributed big data processing frameworks with WAN-bandwidth awareness and provide pros and cons for most of the frameworks. We also categorise them based on the processing technique (batching, micro-batching, native streaming) and what big data framework they are based on (MapReduce, Spark, Flink) and compare them based on several features such as data locality, multi-cluster support and architecture type. We give our recommendations for future work such as the need for privacy, security and authentication mechanisms that are missing in all the frameworks, a decentralised architecture that offers flexibility and fault tolerance in GDA systems and the use of machine learning to improve the scheduling and task placement.

The remainder of this paper is organized as follows. "Background" section introduces and compares big data batch and streaming frameworks that are used by GDA systems surveyed in this paper. "Motivation" section provides the reasons and motivations behind designing geo-distributed big data frameworks. "Challenges" section describes the challenges facing geo-distributed big data processing systems. In "Geo-distributed big data processing" section we review, categorise and compare different bandwidth-aware GDA frameworks. Finally, "Conclusion and open issues" section concludes the paper.

Background

In this section, we briefly introduce the background of big data batch and stream processing frameworks.

Batch-processing frameworks

Batch-processing is a widely used way of processing large amounts of data collected over a period of time. Data collected over a day, week or month, undergoes processing at the end of that period of time for various analytical jobs. Obviously processing large amounts of data takes considerable amount of time (minutes to hours) before getting

any results. The following sections briefly introduce MapReduce and two of the most used frameworks for processing data in batches.

MapReduce

MapReduce is a programming model introduced by Google in 2004 for parallel processing of large datasets on a group of machines in a scale and failure-free manner [7]. As shown in Fig. 2, MapReduce processes data in four phases, the input data is divided into splits and assigned to *mapper* processes each running on a different machine in a distributed system. In the map phase, the splits are processed by applying a user-defined map function and transforming the input data (key/value) into intermediate data that are sorted by keys. In the shuffle phase, the intermediate data is collected by the *reducer* from each mapper. In the reduce phase, the data from the mappers is processed by applying a user-defined reduce function to generate the final output [31].

Apache hadoop

Apache Hadoop is an open source implementation of MapReduce for distributed storage and parallel processing of large datasets on clusters of nodes [8]. MapReduce jobs are submitted to a *resource manager* that supervises and assigns the execution of tasks to *node managers*. The resource manager is responsible for resource allocation, while a node manager monitors the node and reserves resource containers for task execution as illustrated in Fig. 3.

Hadoop Distributed File System (HDFS) is a distributed file system for data storage used by MapReduce applications. In HDFS, the input files for a MapReduce job are divided into blocks (64MB or 128MB) with each block replicated in a set of DataNodes for fault tolerance [31].

Apache spark

Apache Spark is a cluster computing platform based on Hadoop MapReduce and extends the model to support more types of computations such as interactive queries [9]. Unlike

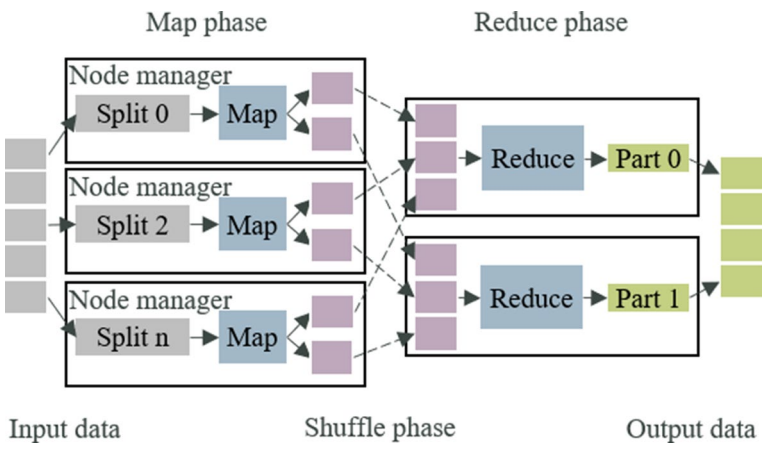


Fig. 2 MapReduce algorithm execution

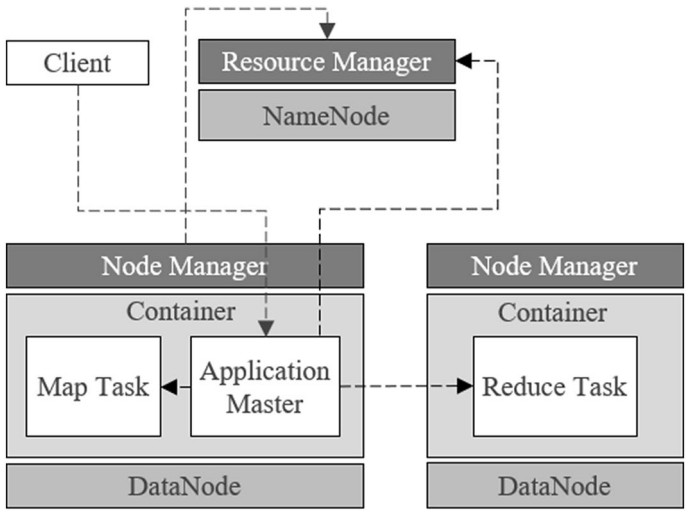


Fig. 3 General architecture of a Hadoop cluster

Hadoop MapReduce that needs to store the outputs of each task on disk, Spark stores the outputs in-memory which increases the speed of data processing.

Resilient Distributed Dataset (RDD) is a fundamental data structure of Spark. Each dataset in RDD is split into logical partitions and can be processed by different nodes. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes [32].

Stream-processing frameworks

Unlike batch-processing, stream-processing is meant to process small size of data immediately and continuously for long period of time (months, years).

There are two types of stream-processing:

- Native streaming: Incoming records are processed immediately, without waiting for others. Operators are processes that run continuously and process all records that pass through them.
- Micro-batching: Incoming records are batched together for a defined small time interval and then processed in a single mini-batch.

Spark streaming

Spark Streaming is an extension of the core Spark API that allow the processing of stream data in micro-batches defined as Discretized Stream (DStream) [33]. DStreams are built on RDDs to perform computations which allow the integration with other Spark components (MLlib, Spark SQL).

Apache flink

Flink is an open source framework that processes stream data as true streams (native streams), records are instantly pipelined through operators as soon as they arrive [34].

Flink provides fault management, high throughput and a compatibility mode that allows the use of existing and unmodified Apache Storm [35] and MapReduce code on the Flink engine.

A comparison of the different processing approaches is given in Table 1 depending on data size, analytic complexity and latency. All these frameworks are designed to process data in a single data centre, where the network is generally available and homogenous.

Motivation

Given the size of existing data centres, some would argue that there is no need for geo-distributed big data analytics. In this section, we list the motivations behind designing geo-distributed big-data frameworks.

Geo-distributed applications support

Organisations operating around the world are deploying applications in geo-distributed data centres to meet customers needs and latency requirements. Thereby, huge volumes of data are generated at these geo-distributed locations and aggregating all the data to a single location for processing has been proven wasteful and costly in terms of resources [11, 12]. Thus, the need for big-data frameworks that can assign computation where data is located and then aggregate only the relevant outputs of these computations at a single site to execute the final computation.

Data centre failure/outrage

A data centre failure is an unusual event to happen; however, when it does happen, it can lead to service interruption for organisations that are hosting applications in that DC. Fault-tolerance is already managed by most big data frameworks for disk, node or rack failure by replicating the data across nodes, but the data replication does not expand outside the data centre. Thus, the need to design systems that introduce new architectures, task, job and data replication techniques (see "Geo-distributed big data processing" section) in order to handle DC failure.

Regulatory constraints

Recently, governments have increased the restrictions on data movement and storage [36], which makes the solution to centrally aggregate all the data to a single location before the computation unattainable. Moving sensitive or confidential raw data (health data) within the country is still acceptable; however, moving sensitive raw data outside

Table 1 Comparison of data processing techniques

Processing Approaches	Batching	Streaming	
		Micro-batching	Native streaming
Data size	Large	Small batch	Small
Analytics	Complex	Simple	Simple
Latency	High(minutes to hours)	Medium	Low
Frameworks	Hadoop, Spark	Spark Streaming, Storm-Trident	Flink, Storm

the country can be a breach to privacy. Therefore, it is mandatory to design geo-distributed processing systems that maintain data privacy by avoiding raw data movement and only transfer the desired intermediate data.

Challenges

Aggregating all the data at one location to be then processed has been proven wasteful, costly and limits the timeliness of the analytics [11, 12]. The better approach is to leave the data "in place" and distribute the tasks of a job across the different clusters. Yet, this approach faces many challenges, as we elaborate in the remainder of this section.

Wide area network constraints

An important characteristic in geo-distributed big data analytics is the network resource heterogeneity. WAN bandwidth is very limited in comparison to Local Area Network (LAN) bandwidth. For example, Zhang et al. [37] report that the intra-DC bandwidth is around 820 Mbps, whereas the inter-DC bandwidth is around 100 Mbps.

In addition, the available bandwidth can considerably differ from one DC to another because of differences in the network hardware and/or traffic of other non-analytics applications running in the same cluster. For example, Viswanathan et al. [38] report the variation among pairs of ten Amazon EC2 regions, as well as between DCs operated by Microsoft. The inter-DC bandwidth fluctuation can lead to unpredictable data transmission time which impacts the performance of geo-distributed computation significantly [39, 40]. Thus, WAN bandwidth is a significant constrain and a bottleneck in geo-distributed big data analytics.

Heterogeneous clusters

As we mentioned in "Introduction" section, most recent works assume that the sites have homogenous and available computational capacities which does not conform to the reality [19]. Clusters are built with different levels of investment and capacity requirements at different times. Hung et al. [19] report that computational resources vary by up to two orders of magnitude across hundreds of sites. Clusters often share resources with non-analytics applications (client services) which limits the available computational resources for data analytics jobs and increase heterogeneity [12, 41].

Furthermore, it has been proven that the memory can become the bottleneck at runtime when running wide-area data analytics queries [42].

Heterogeneous data sizes

In a globally distributed sensor network, the size of the data generated is dependent on the frequency and the number of sensors. Over time the data distribution varies significantly and is not constant. Moreover, for a geo-distributed job, the data needed at different sites to run the analyses may not be the whole distributed dataset. Thus, it is problematic to balance and supply the sites with computational resources proportional to the size of the data generated.

Geo-distributed big data processing

In this section, we survey geo-distributed big data frameworks that consider bandwidth in the scenario where the data is already distributed over multiple sites before computation.

We review several frameworks based on MapReduce, Spark and Flink, and compare them based on several features such as resource management, fault tolerance, data locality and multi-cluster support (see Table 2). We also categorise them under two architectures, as follows:

- Centralised architecture: A single master located at one of the sites controls the resources of all the workers at all the sites, as shown in Fig. 4b.
- Decentralised architecture: Each site has a master that controls its own workers. Each site can run traditional single cluster jobs, and also collaborate and share computational and data resources to support geo-distributed jobs, as shown in Fig. 4a.

Geo-distributed batch processing frameworks

MapReduce-based frameworks

Medusa is a platform based on MapReduce that allows geo-distributed computation without any modification to the Hadoop framework and can deal with three faulty scenarios: accidental, malicious, and cloud outages [43]. Medusa starts f+1 replicas of a MapReduce job in different clouds. It validates the computation by comparing the outputs of the replicated jobs and deals with accidental faults by re-executing the faulty job in the same clouds. For malicious faults or cloud outages, the system re-executes the faulty job in another cloud. Medusa can identify the compromised cloud in the scenario of a malicious fault.

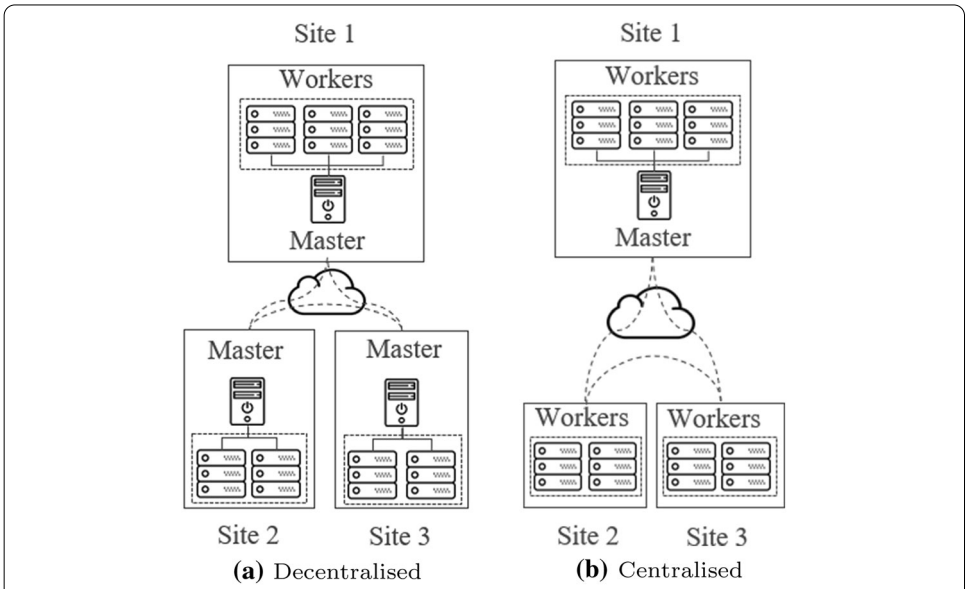


Fig. 4 Decentralised vs. centralised architecture

Table 2 Comparison of data processing techniques

Frameworks	Resource management	Data locality	Multi-query support	Cluster architecture	Multi-cluster support	Relevant intermediate data finding	Fault-tolerance support	Heterogeneous resources	Inter-DC traffic reduction	Machine learning use	Cost-awareness
Batch processing MapReduce-based frameworks											
Medusa [43]				C			✓				
Chrysaor [44]				C			✓				
GeoDis [45]		✓		C				✓			
Li et al. (1) [46]		✓		C					✓		
Batch processing Spark-based frameworks											
LinePro [47]		✓		C							
Houtu [37]	✓		✓	D	✓		✓	✓			
Tetrium [19]	✓		✓	C		✓		✓	✓		
Lube [42]		✓		C				✓		✓	
Harmony [48]		✓		C				✓	✓		
Kimchi [49]				C			✓	✓	✓		✓
Other batch processing frameworks											
HDM-MC [50]				D	✓			✓			
Geo-distributed SQL-style processing systems											
Turbo [51]		✓		C		✓			✓	✓	
Bohr [52]		✓		C		✓			✓		
Micro-batch processing frameworks											
Iridium [12]		✓		C		✓			✓		
Li et al. (2) [53]			✓	C					✓	✓	
Native stream processing frameworks											
Sana [54]			✓	C		✓	✓		✓		
WASP [55]	✓		✓	C			✓	✓	✓		
JetStream [56]		✓		C			✓		✓		

C Centralised architecture, *D* Decentralised architecture

Chrysaor is similar to Medusa but replicates tasks instead of jobs [44]. A proxy compares every replica of the output of all map and reduce tasks to identify faulty tasks and immediately re-execute them, instead of waiting until the end of the job execution. Chrysaor launches f+1 replicas of the faulty task in the same cloud when dealing with accidental faults. For malicious faults or cloud outages, Chrysaor executes f+1 replicas of the faulty task in another cloud when dealing with a fault in a map task. Chrysaor needs to re-execute the full job if a malicious fault or cloud outage happens during the reduce phase.

Pros: Both Medusa and Chrysaor schedule the replicated tasks across multiple clouds [57], the best cloud (for Medusa) or clouds (for Chrysaor) based on computational power (number of cores of CPU, clock speed, total memory) and bandwidth which decreases the job completion time.

GeoDis is the first locality and network aware scheduler to consider both data locality (with replication) and data migration together for optimising the makespan of data-intensive jobs on geo-distributed systems [45]. The centralised global scheduler manages a First-In First-Out (FIFO) queue for all submitted jobs and decides where to place tasks and which replica to access. In each DC a local scheduler maintains and reports to the global scheduler the progress of the local task queue (see Fig. 5). The task placement and the data access problem is described as a Linear Program (LP) and solved using the GLPK solver [58]. The proposed online heuristic algorithm favours data locality to data migration when it is possible but in the case where the data needs to be transferred from a remote data centre, the algorithm will select the data replica from the data centre with the fastest link.

GeoDis can decrease the makespan of processing jobs by 44% as compared to the state-of-the-art algorithms and remain within 91% closer to the optimal solution by the LP solver [45].

Li et al. (1) [46] proposed a geo-distributed MapReduce framework that minimises the traffic between DCs by jointly considering input data movement and task placement. The input data can be moved by map tasks running at remote DCs if the total inter-DC traffic is reduced. The data movement and task placement problem is formulated as a non-linear optimisation problem and solved using a linearisation technique to replace the non-linear constraints with linear ones. An approximation approach by relaxing one of the constraints is used to achieve predicted job completion time.

The system design has three main components:

- Parameter extractor: Estimates parameters such as the bandwidth between clusters and information about the OI-ratio by analysing the execution history of similar jobs, then sends the parameters to the optimiser.
- Optimiser: Runs an algorithm that determines input data movement and task placement based on the estimated parameters. The algorithm minimises the inter-cluster traffic incurred by the MapReduce job.
- Data loader and task assigner: The input data is retrieved by the data loader according to the task placement choices made by the optimiser. For each data split loaded, the task assigner starts a map task and some reduce tasks.

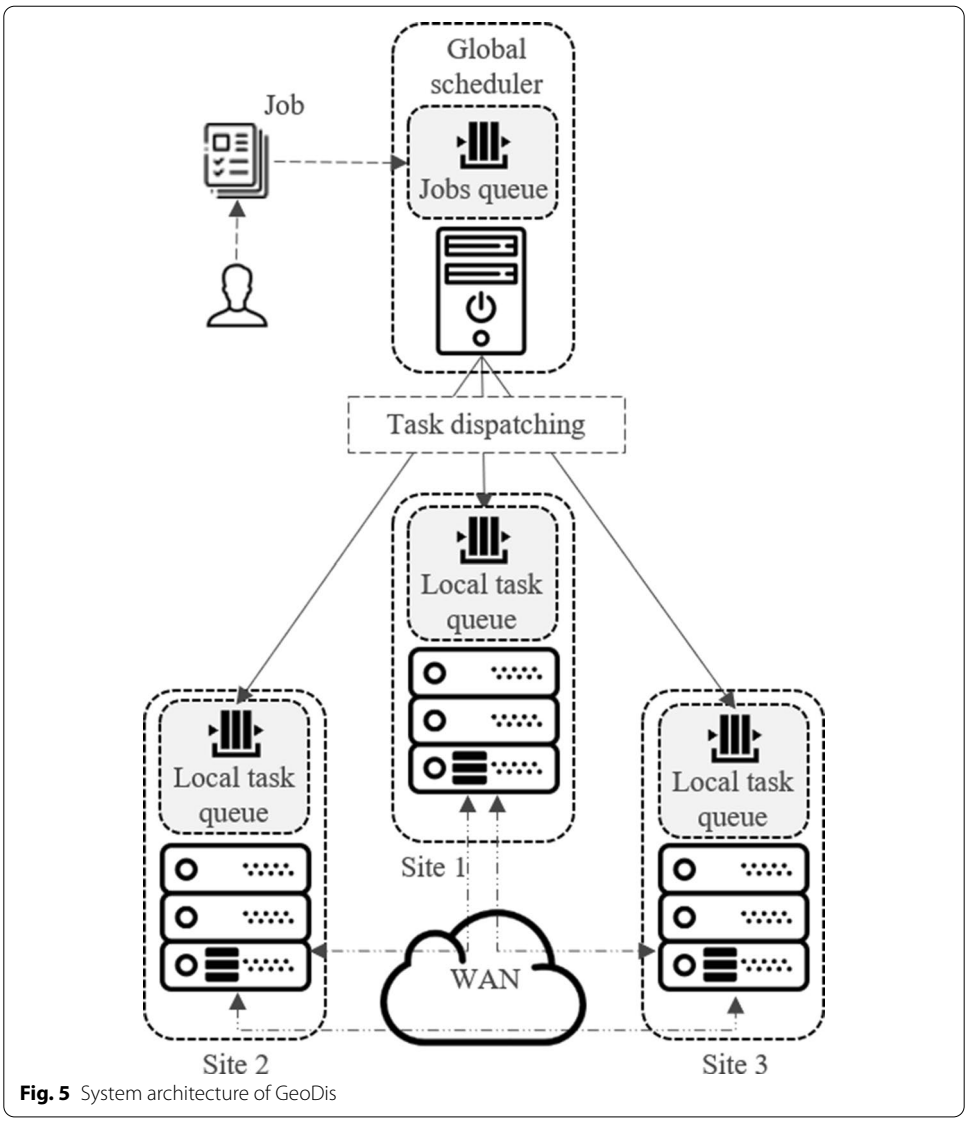


Fig. 5 System architecture of GeoDis

Spark-based frameworks

Lube is a geo-distributed framework that reduces the query response times by detecting bottlenecks at runtime [42]. Lube monitors the performance metrics (CPU, memory, network and disk) in real-time and uses Autoregressive Integrated Moving Average (ARIMA) [59] or the Sliding Hidden Markov Model (SlidHMM) [60] to detect resource bottleneck at runtime. The scheduling algorithm considers data locality and bottleneck severity to assign tasks to worker nodes, the late-binding algorithm in Sparrow [61] is used to avoid false positives when detecting bottlenecks by holding a task for a short time before submitting it to a worker node.

Lube is the first framework that uses machine learning techniques to detect runtime bottlenecks. Lube is implemented on Spark and achieves 90% accuracy for bottleneck detection and reduces the query response time by 33% when deployed across 9 Amazon EC2 regions [42].

Pros: The authors claim that Lube is the first work that uses machine learning to detect runtime bottlenecks.

Cons: The authors did not mention any dataset for training or testing the machine learning model.

LinePro is a data and network aware algorithm on top of Spark for geo-distributed big data processing [47]. The algorithm takes advantage of the data locality to reduce the transfer cost by scheduling the computations which are ready to be executed rather than scheduling all the computation at the same time. The reduce computation movement problem is described as an Integer Linear Program (ILP) and solved using the Gurobi solver [62]. The main components of the computation movement model are:

- TaskDefinition: an object containing the following attributes: stageId, shuffleId, taskbinary, parts, locations and cost matrix. The cost matrix is used to produce the final locations of a reduce task. Locations contain the nodes where the computation can be moved to. shuffleid defines whether the task belongs to the reduce phase or not.
- MapOutTracker: tracks and provides the data locations and the output size of the map tasks to DAGScheduler.
- DAGScheduler: this function submits the missing task sets which need to be executed and use MapOutTracker and RDD dependency to produce the TaskDefinition for each task.
- TaskSchedulerImpl: calculates the computation location. The map computation location is provided by the RDD dependency while the reduce computation location is provided by the ILP through the cost matrix and TaskDefinition.
- TasksetManager: when each node is overloaded and cannot execute a task set in one stage, the cost matrix is used to build a sorted cost list for each node. The node can pick the smallest cost from the cost task list when compute resources are available.

Reportedly, LinePro improves the performance of geo-distributed data processing by 22% as compared to the Spark default scheduler when deployed on 3 data centres [47].

Cons: LinePro performs poorly if the data set distribution is unbalanced between data centres.

Tetrium is a geo-distributed system for multiple resource allocation designed on top of Spark, that considers heterogeneous bandwidths and compute resources to schedule jobs, order and place map and reduce tasks [19]. Tetrium consists of two managers, as follows.

- Global manager: located in only one site for coordinating analytics jobs across sites, adjusting the workloads, and keeping track of data locations.
- Site manager: located at each site for executing assigned jobs, controlling local resources and notifying the global manager of resources drop changes.

To schedule multiple and competing jobs over a geo-distributed cluster, the problem is simplified by first solving the scheduling of a single job then extending the solution

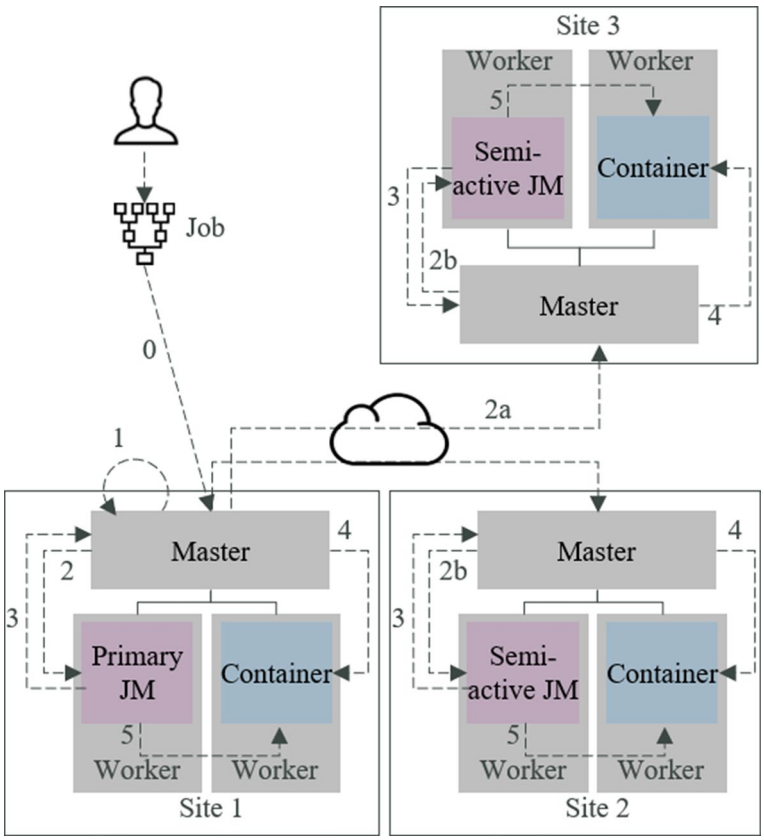


Fig. 6 A job lifecycle in Houtu

to multiple jobs. A Linear Program is described to optimise the execution time of a single job while considering the available compute slots and bandwidth as constraints. Then the Shortest Remaining Processing Time (SRPT) uses the solution given by the LP from every job to schedule and order competing geo-distributed jobs [19].

Tetrium [19] improves the average job response time by up to 78% compared to existing locality techniques [9, 63], up to 55% compared to Iridium [12], and 33% compared to Tetris [64] when deployed across eight Amazon EC2 regions in five continents running the TPC-DS [65] and the Big Data benchmarks [66].

Pros: Tetrium is the first effort towards multi-resource scheduling for data analytics jobs across geo-distributed clusters that considers data distribution, compute capacities and network bandwidths heterogeneity.

Cons: Tetrium does not consider network congestion among DCs and is not suitable for stream-oriented workloads.

Houtu is a decentralised geo-distributed data analytics system that is designed to efficiently operate over a collection of DCs [37]. Each DC can run jobs in a traditional single cluster mode as well as in a geo-distributed mode by collaborating with other DCs. Houtu applies an adaptive feedback algorithm (AF) and parametrized delay scheduling with work stealing (Parades) that extends the delay scheduling algorithm [63] in each Job Manager (JM) to respectively manage resources and schedule tasks for the geo-distributed job. A job's life cycle in Houtu consists of 6 steps, as illustrated in Fig. 6.

- Job submission and job manager generations:

  (Step 0) A user submits a job in the form of a directed acyclic graph (DAG) to the master. (Step 1) The master generates job managers according to the job description. (Step 2) A primary JM is created directly in its own cluster. (Step 2a) The job description is sent to the remote masters if remote resources are needed. (Step 2b) Semi-active JMs are created in the remote clusters.
- Resource request and task executions:
  (Step 3) The job managers send requests to their local masters for computational resources (task executors/containers). (Step 4) The masters (job schedulers) schedule resources to the JMs by returning containers for the requested resources. (Step 5) JMs submit the tasks that run in the containers. Steps 3-5 are repeated as DAG jobs unfold in multiple waves.

Houtu is built in Spark [32] on a YARN [67] system; while Zookeeper [68] is used to synchronize the JMs for the same job. When deployed in 20 machines across 4 regions on Alibaba Cloud (AliCloud), Houtu has 29% improvement in terms of average job response time, and 31% improvement in terms of makespan in comparison to the decentralised architecture with a static scheduling algorithm.

Pros: The decentralised architecture proposed in Houtu provides the flexibility of single and multi-cluster jobs while respecting regulatory constraints.

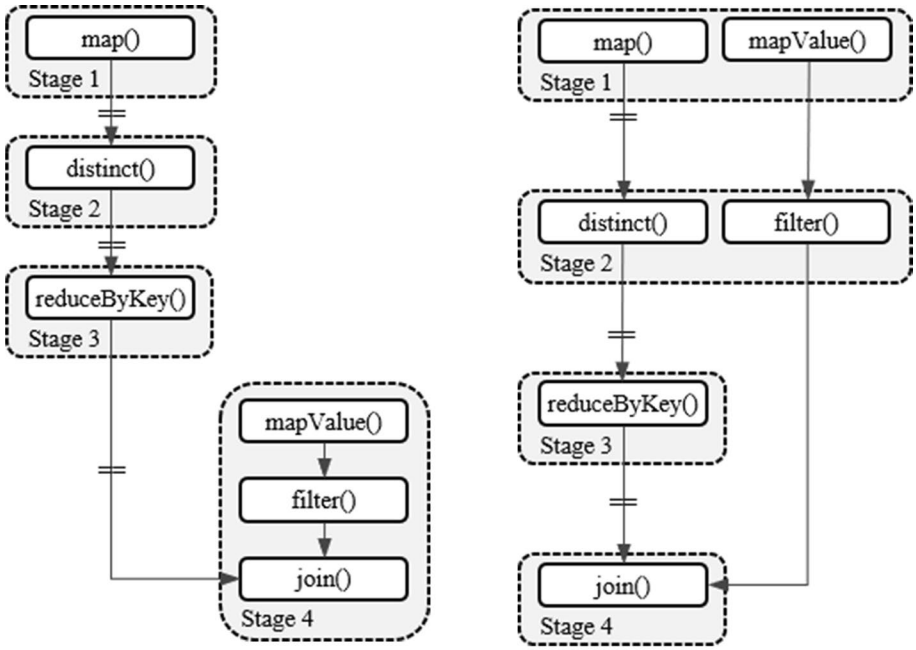
Kimchi is a cost-aware geo-distributed data analytics system that determines *reduce task placement* by considering data transfer cost, WAN bandwidth, intermediate data size and locations as well as the preferences for the trade-off between cost and performance of applications [49].

Kimchi is the first GDA system that optimises reduce task placement while considering multi-cloud data transfer cost i.e., the cloud providers have different data transfer rates.

The task placement is described as a MIP that takes the following inputs:

- Trade-off Preference: Applications need to provide a number between 0 and 1, 0 being the minimum cost and 1 being the minimum latency.
- Data Transfer Cost: This information is gathered from the websites of cloud providers.
- Network Bandwidth Information: Executors on each DC estimate the bandwidth when data is transferred between DCs.
- Data Size for Shuffle Tasks: This information is available from the MapOutputTracker in Spark.

The output of the MIP is a set of pairs (task, DC) plus the expected latency and network cost of each task. The MIP is called at run time of each shuffle stage, then the output is provided to the scheduler that assigns tasks to DCs. If dynamics (network contention and bandwidth changes) are detected, then the scheduler calls a heuristic that assigns the task to another idle DC while trying to respect the trade-off preference.



(a) Spark's Staging Strategy **(b)** Harmony's Staging Strategy

Fig. 7 Program DAG with Different Staging Strategies

Kimchi [49] is built on the Spark framework [32]. It supports new Spark properties that control Kimchi's settings. Kimchi reportedly reduces cost by 14–24% without impacting performance and reduces query execution time by 45–70% without impacting cost compared to other baseline approaches centralised (minimum cost), vanilla Spark, and bandwidth-aware (e.g., Iridium [12]).

Pros: Kimchi offers great flexibility; applications can choose between the best latency regardless of cost and minimum cost regardless of latency.

Harmony is a geo-distributed processing framework that jointly considers WAN-bandwidth and computational capacity for staging and scheduling with the goal of minimising application execution time [48].

In Apache Spark a program is defined as a DAG the nodes and edges of which represent operators and data dependencies, respectively. The Spark staging strategy uses shuffle dependency to order operators into stages which prevent some operators from being executed in previous stages as shown in Fig. 7a.

Harmony's staging strategy starts by determining the critical path in the DAG which is a set of operator nodes that provide the shortest execution time. A greedy algorithm determines the starting and finishing time of each operator and groups them based on shuffle dependency as illustrated in Fig. 7b. Operators who are not shuffle-dependent are assigned to stages depending on their start and finish times.

To minimise the overall computation time, the Harmony scheduler determines the minimal computation time of each stage using the input data size and the computation capacity of each DC, and then the data transfer plan is computed based on DC's up-link and down-link. Before assigning tasks to each location, the input data

is re-allocated to different DCs to meet the minimal computation stage time and the minimum data transfer cost based on the data transfer plan.

Harmony [48] is implemented on Apache Spark and is reported to be 1.6 and 2.1 times faster than Iridium when deployed over five AWS EC2 locations with uniform and non-uniform network link bandwidths, respectively.

SQL-style processing frameworks

Turbo is a geo-distributed analytics framework that uses machine learning to predict the output size and the execution time of JOIN queries in order to optimise the performance of geo-distributed queries by dynamically changing the query executing plan in response to resources variations such as memory, CPU and WAN bandwidth [51].

The framework is designed to perform on top of current frameworks like Spark, Hive and Pig without altering the lower layer functionalities such as task scheduling/placement and data replication.

A 15k samples dataset is built by running queries from TPC-H benchmark and recording the output size and the completion time of each query and also features like CPU core number, memory size, available bandwidth, etc. Handcrafted features and features crossing are used to include nonlinear features that might help in predicting the completion time and output size of a query.

The architecture of Turbo consists of three components, as illustrated in Fig. 8:

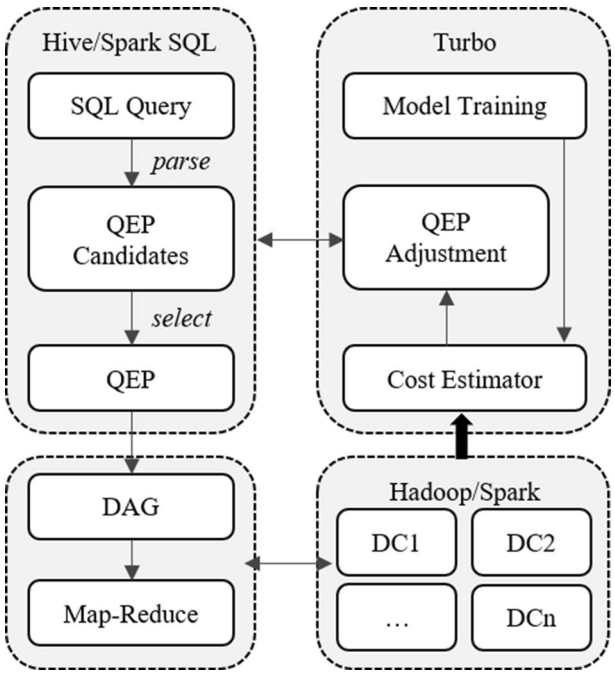


Fig. 8 Overview of Turbo

- Model training: Least Absolute Shrinkage and Selection Operator (LASSO) is a regression analysis method used to select the most important features and discard irrelevant ones for predicting the output size. LASSO alone is unable to predict the completion time as the relationship between the selected features and the completion time is nonlinear. Gradient Boosting Regression Tree (GBRT) is a regression technique that produces a strong prediction model from an ensemble of weak regression trees. Unlike linear models, the boosted trees model are able to capture non-linear interaction between the features and the target. The GBRT takes the features selected by LASSO as an input to predict the completion time.
- Cost estimator: A query execution plan may contain a series of joins, the output of a join is the input of the next join. In the scenario when the output is located only in a DC, the cost is predicted by the ML model, but when the reduce tasks are placed on multiple DCs, the output of the current join is spread across those DCs and the cost can not be predicted by the ML model, In this case the cost is predicted by a divide and conquer heuristic that partition the join in a series of sub joins that can be predicted by the ML model.
- Runtime QEP adjustment: Regularly adapts the query execution plans to runtime dynamics by exploring three greedy policies to choose the next pairwise join with the least lookahead cost [51]:

  - Shortest Completion Time First (SCTF) selects the next pairwise to be the one that is expected to have the least completion time.
  - Maximum Data Reduction First (MDRF) selects the next pairwise to be the one that is expected to result in the greatest difference in volume between input and output data.
  - Maximum Data Reduction Rate First (MDRRF) selects the next pairwise to be the one that is expected to maximise the data reduction, that is the total input size minus the output size divided by the estimated joint completion period.

Turbo achieves a cost estimation accuracy of over 95% and reduces the query completion times by up to 40% when deployed on a Google Cloud Cluster with 33 instances distributed across 8 regions [51].

Bohr is a similarity aware geo-distributed data analytics system that reduces the shuffle time and consequently minimises the query completion time [52]. Bohr pre-processes the generated data by storing it in OLAP data cubes [69], and then when a query arrives for the first time, Bohr uses OLAP instructions (dice, slice or roll-up) to retrieve the attributes needed for the query and run similarity search [58] based on these attributes to organise the data. This prepares the datasets for similarity-aware data placement when a query recurs.

Bohr extends Iridium's task and data placement [12] by using a probe that contains the top-k representative records of its dataset to identify data to be moved from the bottleneck DC to other DCs. A linear program (LP) is used for the reduce task placement and bottleneck DCs detection.

Bohr is based on Spark and can reduce the query completion time by 30% in comparison to Iridium when deployed across ten Amazon EC2 regions [52].

Pros: Bohr further extends Iridium [12] by using data similarity to optimise which data should be moved rather than moving datasets with many queries accessing them.

Cons: Similar to Iridium [12], Bohr makes assumptions on query arrivals and adds overhead by using OLAP cubes.

Other batch processing frameworks

Hierarchically Distributed data matrix-multi cluster (HDM-MC) is a big data processing framework that can run large scale data analytics over single or multiple clusters [50]. HDM-MC is an extension of the Hierarchically Distributed Data Matrix (HDM) [70] which is a data representation (that contains format, locations, dependencies between input and output) designed to support parallel execution of data-intensive applications. The framework consists of three main components, which are responsible for multi-cluster coordination, planning and scheduling, respectively. We review each of them in detail below.

- Multi-cluster coordination. HDM-MC supports two types of coordination architecture: hierarchical architecture and decentralised architecture. In the hierarchical architecture, there are one or more super-master clusters that coordinate multiple child-master clusters, each of which contains workers/resources. In the decentralised architecture, there are no super-masters; each master has two to three sibling-masters, they can collaborate with, while managing their own workers and updating information about resources that can be used by their siblings.
- Multi-cluster planning. The planning phase is performed in two steps: stage planning and task planning. At the stage planning step, a computational job that needs to be executed is divided into multiple job stages, each of which belongs to one of the following job categories:

  – Local: all the input datasets are in the cluster that performs the job planning. This job stage is scheduled in the current cluster.
  – Remote: all the input datasets are in another cluster. This job stage is submitted for execution at the cluster that contains the input datasets.
  – Collaborative: the input datasets are distributed among multiple clusters. This job stage is planned to be parallelised and scheduled on both the current and sibling clusters.

  Then at the task planning step, each job stage identified in the stage planning step is scheduled to be executed in one of the master clusters that break down the job into tasks for scheduling.

- Multi-clusters scheduling. After a job is explained, it goes through a two-layer scheduling process. The first layer monitors and schedules the stages of each job, while the second layer receives, monitors and schedules the tasks of each active job stage by applying one of the following three scheduling strategies:

- Delay Scheduling: Arriving tasks will wait for a short duration of time in order to achieve better data locality.
- Min/Max Scheduling: Tasks are scheduled based on the estimated minimum completion time. Min/Max is aware about the distance between workers in the network.
- Hungarian Algorithm. It is a graph algorithm that finds the near optimal shortest distances among the nodes (workers) of a graph.

Pros: HDM-MC, like Houtu, provides dynamic switching of architecture between single and multi-clusters but is not based on any of the existing big data frameworks.

Geo-distributed stream processing frameworks

Micro-batch processing frameworks

Iridium is a low latency geo-distributed analytics system that minimises query response times by optimising data and tasks placement of the queries [12]. The system redistributes datasets between DCs prior to queries' arrivals and places the tasks in sites with better bandwidth to reduce network bottlenecks during the execution. The task placement problem is formulated as a Linear Program LP that models the site bandwidths and query characteristics, and it is solved using the Gurobi solver [62]. A greedy heuristic iteratively moves small chunks of the most accessed datasets and/or datasets that produce large amount of intermediate data. The architecture of Iridium consists of two main components, a local manager and a centralised global manager.

- Local manager: Executes assigned tasks and keeps track of the available resources at each site.
- Centralised global manager: Converts queries into directed acyclic graphs (DAGs) of stages, coordinates query execution and keeps track of data locations across sites.

Iridium [12] is implemented on top of Spark and reportedly speeds up queries by 64% to 92% as compared to Conviva [71], Bing Edge, TPC-DS [65] and Berkeley Big Data Benchmark [66]. It also saves WAN bandwidth usage by 15% to 64%, when deployed across eight Amazon EC2 regions [12].

Pros: Iridium incorporate a "Knob" for budgeted WAN usage, and minimises query execution latency by finding and moving relevant intermediate data prior to the arrivals of queries based on history.

Cons: Iridium does not consider network congestion among DCs and makes assumption on query arrivals.

Li et al. (2) [53] proposed a geo-distributed Spark based streaming framework that aims at reducing the processing time of each micro-batch by jointly considering micro-batch sizing, bandwidth, task scheduling and routing of data streams. The problem is described as a non-convex optimisation problem and solved with a combination of the Alternating Direction Method of Multipliers (ADMM) [72] and LASSO.

The proposed ADMM algorithm aims at reducing query response time by selecting the fastest path (i.e., path with the highest bandwidth) to route Spark DStream from

each source to its collection site. The algorithm can converge within three to four iterations. At each iteration, the batch size gets smaller which leads to a selection tree that has more available bandwidth on its links (a smaller batch size means that the query is executed frequently which lead to more network flows), whereas a tree with less bandwidth will force the system to choose a larger batch size.

The proposed Spark based streaming framework reportedly reduces query processing latency and improves network transfer times compared to the original Spark Streaming framework when deployed over a cluster of seven Amazon EC2 instances with an emulated bandwidth running coexisting queries of different types (including WordCount, Grep and Top-k) [53].

Native stream processing frameworks

JetStream is a geo-distributed stream processing system that optimises the processing through adaptive sampling and data cube abstraction [56]. The system architecture has three main components:workers on each node for data processing, a centralised coordinator manages and distributes computation across available workers, and a client.

The life cycle of a query in JetStream begins once a client program generates and sends a data-flow graph for execution. The data-flow graph is then checked for type and structural errors and submitted to the coordinator. The centralised coordinator starts assigning linked data-flow operators to workers then sends the relevant subset of the graph to each node. The nodes establish network connections between each other and start the operators. The query stops running when the coordinator sends a stop signal or all the sources send a stop message indicating that there will be no more data.

Pros: JetStream deals with the WAN bandwidth limitation by making a compromise between the quality of the final results and performance, which is good for small sensor networks.

Cons: The trade-off between the output quality and performance provides inaccuracy in the final results.

Sana is a WAN Aware geo-distributed stream processing system based on Apache Flink that incrementally enables geo-distributed queries to share their common execution [54].

The system uses three types of sharing policies: (1) input-sharing: queries share a common subset of input data, (2) operator-sharing queries share the same execution/data processing on the same input data, (3) output-sharing: queries share parts of the output or intermediate results [54]. The system utilises the sharing policies to generate the query execution plans and to schedule the execution in a WAN-Aware manner.

The system design consists of five main components, as illustrated in Fig. 9:

- WAN Monitor: monitors the WAN bandwidth availability between sites and detects congested links through the ratio of the current bandwidth utilisation over the maximum available bandwidth.
- Shared Job Manager: keeps track of the deployment of existing queries and provides this information to the query optimiser.

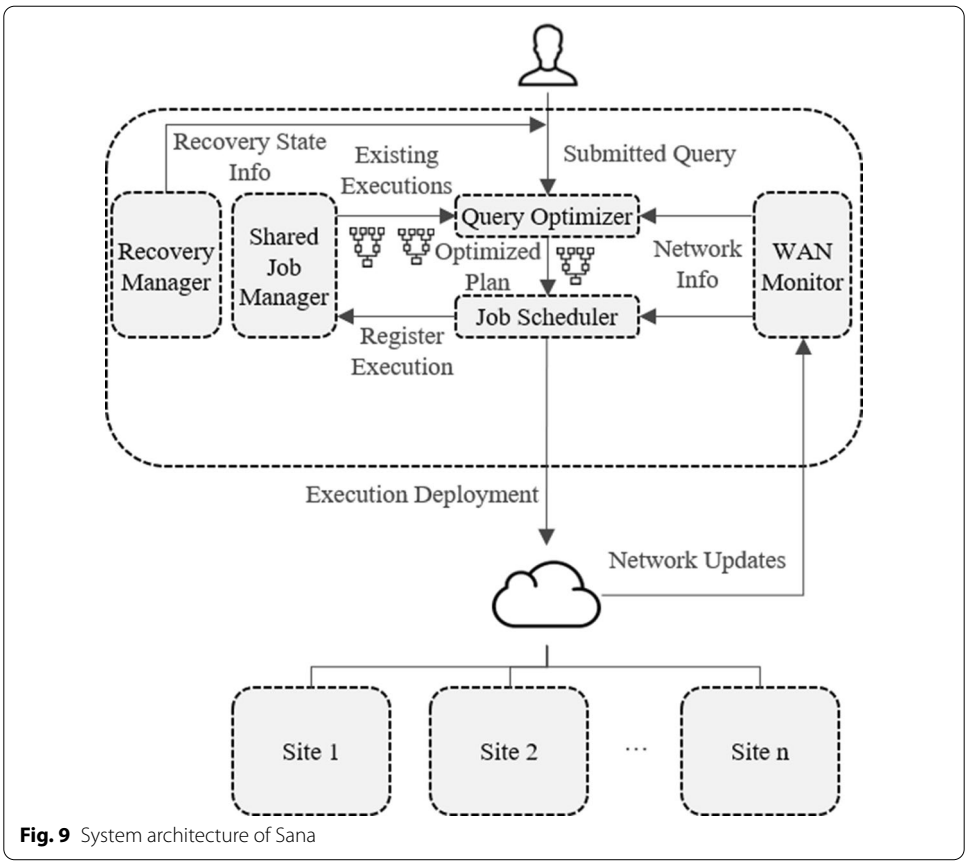


Fig. 9 System architecture of Sana

- Query Optimiser: optimises and generates the query execution plan by identifying commonalities between new queries and existing ones and considering the inter-site bandwidth information.
- Job Scheduler: schedules and deploys each operator instance on a compute node while minimising the latency and/or WAN bandwidth consumption.
- Recovery Manager: keeps track of the query execution state and allows the system in the case of failures to relaunch a query from its last execution state.

Sana achieves 21% higher throughput while saving WAN bandwidth utilisation by 33% [54] when deployed across 14 geo-distributed Amazon Elastic Compute Cloud (EC2) data centres running on real Twitter data that was collected from Twitter Streaming APIs [73].

WASP is a resource-aware Wide-Area Adaptive Stream Processing system that aims at maintaining low latency execution when dynamics occur (resource or workload variation, stragglers, failures) by adjusting the physical or logical plan of queries at runtime using multiple techniques such as task re-assignment, operator scaling, and query replanning [55]. WASP system architecture consists of a Job Manager (JM) and multiple geo-distributed Task Managers (TM) in each DC.

The JM includes:

- Query planner: Generates query execution plans.
- Scheduler: Schedules and assigns tasks to TMs.
- WAN monitor: Monitors and reports WAN bandwidth to the Query Planner and the Scheduler.
- Global Metric Monitor: Gathers metrics from each TM through a Local Metric Monitor and identifies dynamics occurrence.
- Reconfiguration Manager: Re-adjusts query execution plan if the Global Metric Monitor detects dynamics.

The TM includes:

- Local Metric Monitor: Monitors and gathers task performance such as processing latency and I/O stream rates.
- Checkpoint Manager: Keeps track of task state to allow tasks to start/resume executions from the last check-pointed state in case of task failure or migration.

The Reconfiguration Manager, the Query Planner and the Scheduler work together to re-adjusts the query execution plan by using one of the following techniques:

- Task re-assignment: The problem of re-assigning tasks is described as an ILP problem that aims at minimising the data streams network transmission delay with inbound and outbound bandwidth as well as computing resources as constraints.
- Operator scaling:
  - Scale up is used to instantiate new operators in site when computational bottleneck is detected.
  - Scale out is used to instantiate new operators across sites when network bottleneck is detected.
  - Scale down is used to reduce the number of operators when resources are over-allocated due to scale up/out or misconfiguration.
- Query re-planning: The Query Planner uses a heuristic cost-based algorithm to generate multiple execution plans, while the Scheduler computes the best task placement for each plan and selects the lowest plan/placement pair delay.

WASP [55] is implemented on Apache Flink [34] and reportedly handles wide-area dynamics with low overhead while maintaining the quality of the results when deployed across eight edge nodes and eight DC nodes using simulated network bandwidth and latency.

Conclusion and open issues

MapReduce, Spark and Flink are widely used for commercial applications and scientific research but are not designed for geo-distributed data analytics. While there are some solution that can run analytics across geo-distributed sites, none of them consider WAN-bandwidth in their solutions, which we believe is one of the most important

factors in geo-distributed big data processing. In this work, we provided an overview of the most used frameworks for big-data analytics and discussed challenges in designing efficient geo-distributed data processing systems. We also investigated new systems that are able to run geo-distributed analytics, while dealing with heterogeneity and considering the WAN-bandwidth.

Based on this survey, we can highlight the following issues and directions for future development:

- Security and privacy: While reviewing geo-distributed big data frameworks we noticed that none of them deal with security and privacy. Due to regulations on data storage and movement imposed by governments, we recommend researchers to focus on designing authentication mechanisms and trust models to make geo-distributed data analytics applicable and realistic.
- Decentralised architecture: Another very important issue that we noticed throughout this survey is the lack of frameworks that support decentralised architecture and multi-clusters. Houtu and HDM-MC are the only systems that support such features. We believe that a decentralised geo-distributed big data system can offer great flexibility in deployment as it provides autonomous geo-distributed clusters that can coordinate for geo-distributed jobs. Such system can avoid a general outrage of the whole system in the case of DC failure. Moreover, a decentralised system combined with a security/authentication mechanism can deal with the regulatory constraints and restrictions.
- *Machine learning:* Lube and Turbo are the first to make use of machine learning to help in efficient scheduling of tasks by bottleneck detection and predicting the time cost of queries. However, some questions are left unanswered. What are the performance metrics to choose for training a bottleneck detection model? What are the best features for accurately and efficiently predict a task or a job execution time? We also noticed that the benchmark used to generate the training dataset and to evaluate Turbo only uses structured data, thus we recommand future research to focus on semi-structured and unstructured data to train and evaluate their systems.
- Resource Manager: The systems reviewed do not use a Resource Manager such as YARN. LAN bandwidth has been added to YARN as a resource recently [74] but to the best of our knowledge none of the existing geo-distributed systems manages WAN-bandwidth as a resource. A resource manager that support WAN bandwidth can be less challenging to design when combined with a decentralised architecture because the system is able to distinguish between local jobs and geo-distributed jobs.

Abbreviations

WAN: Wide Area Network; DC: Data Centre; HDFS: Hadoop Distributed File System; RDD: Resilient Distributed Dataset; LAN: Local Area Network; LP: Linear Program; ARIMA: Autoregressive Integrated Moving Average; SlidHMM: Sliding Hidden Markov Model; ILP: Integer Linear Program; SRPT: Shortest Remaining Processing Time; JM: Job Manager; DAG: Directed Acyclic Graph; LASSO: Least Absolute Shrinkage and Selection Operator; GBRT: Gradient Boosting Regression Tree; ML: Machine Learning; SCTF: Shortest Completion Time First; MDRF: Maximum Data Reduction First; MDRRF: Maximum Data Reduction Rate First; OLAP: Online Analytical Processing; HDM-MC: Hierarchically Distributed Data Matrix -Multi Cluster.

Acknowledgements
The authors thank the anonymous reviewers for their helpful suggestions and comments.

Authors' contributions
All mentioned authors contribute in the elaboration of the paper. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Consent for publication
Not applicable.

Author details
[1] Laboratory of Intelligent Systems and Applications, Department of Computer Science, Faculty of Sciences and Technologies, University of Sidi Mohammed Ben Abdellah, Fez, Morocco. [2] Department of Computer Science and Information Systems, University of Limerick, Limerick, Ireland.

Received: 18 November 2020 Accepted: 15 February 2021
Published online: 25 February 2021

References

1. Tudoran R, Antoniu G, Bougé L. SAGE: geo-distributed streaming data analysis in clouds. In: 2013 IEEEinternational symposium on parallel distributed processing, workshops and Phd Forum. 2013, vol. 2013, p. 2278–81.
2. Tudoran R, Costan A, Wang R, Bougé L, Bridging Antoniu G. Data in the clouds: an environment-aware system for geographically distributed data transfers. In: 2014 14th IEEE/ACM international symposium on cluster, cloud and grid computing; 2014, p. 92–101.
3. Cardosa M, Wang C, Nangia A, Chandra A, Weissman J. Exploring mapreduce efficiency with highly-distributed data. In: Proceedings of the second international workshop on mapreduce and its applications. ACM; 2011. p. 27–34.
4. Heintz B, Chandra A, Sitaraman RK, Weissman J. End-to-end optimization for geo-distributed mapreduce. IEEE Trans Cloud Comput. 2016;4(3):293–306.
5. Rabkin A, Arye M, Sen S, Pai V, Freedman MJ. Making every bit count in wide-area analytics. In: Presented as part of the 14th workshop on hot topics in operating systems. USENIX; 2013.
6. Wang L, Tao J, Ranjan R, Marten H, Streit A, Chen J, et al. G-Hadoop: mapreduce across distributed data centers for data-intensive computing. Fut Gener Comput Syst. 2013;29(3):739–50.
7. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.
8. Apache Hadoop. http://hadoop.apache.org/.
9. Apache Spark. http://spark.apache.org/.
10. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: distributed data-parallel programs from sequential building blocks. SIGOPS Oper Syst Rev. 2007;41(3):59–72.
11. Vulimiri A, Curino C, Godfrey PB, Jungblut T, Karanasos K, Padhye J, et al. WANalytics: geo-distributed analytics for a data intensive world. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. SIGMOD '15. ACM; 2015. p. 1087–1092.
12. Pu Q, Ananthanarayanan G, Bodik P, Kandula S, Akella A, Bahl P, et al. Low latency geo-distributed data analytics. SIGCOMM Comput Commun Rev. 2015;45(4):421–34.
13. Jayalath C, Stephen J, Eugster P. From the cloud to the atmosphere: running mapreduce across data centers. IEEE Trans Comput. 2014;63(1):74–87.
14. Jonathan A, Ryden M, Oh K, Chandra A, Weissman J. Nebula: distributed edge cloud for data intensive computing. IEEE Trans Parallel Distrib Syst. 2017;28(11):3229–42.
15. Kettimuthu R, Agrawal G, Sadayappan P, Foster I. Differentiated scheduling of response-critical and best-effort wide-area data transfers. In: 2016 IEEE international parallel and distributed processing symposium (IPDPS); 2016. p. 1113–22.
16. Hu Z, Li B, Luo J. Flutter: scheduling tasks closer to data across geo-distributed datacenters. In: IEEE INFOCOM 2016—The 35th Annual IEEE international conference on computer communications; 2016. p. 1–9.
17. Sundaresan S, de Donato W, Feamster N, Teixeira R, Crawford S, Pescapè A. Broadband internet performance: a view from the gateway. SIGCOMM Comput Commun Rev. 2011;41(4):134–45.
18. Sitaraman RK, Kasbekar M, Lichtenstein W, Jain M. 16. In: Overlay networks: an akamai perspective. Wiley; 2014. p. 305–28.
19. Hung CC, Ananthanarayanan G, Golubchik L, Yu M, Zhang M. Wide-area Analytics with Multiple Resources. In: Proceedings of the thirteenth eurosys conference. EuroSys '18. ACM; 2018. p. 12:1–12:16.
20. Zhou AC, Ibrahim S, He B. On achieving efficient data transfer for graph processing in geo-distributed datacenters. In: 2017 IEEE 37th international conference on distributed computing systems (ICDCS); 2017. p. 1397–1407.

21. Vulimiri A, Curino C, Godfrey PB, Jungblut T, Karanasos K, Padhye J, et al. WANalytics: geo-distributed analytics for a data intensive world. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. SIGMOD '15. New York, NY, USA: Association for Computing Machinery; 2015. p. 1087–92.
22. Kloudas K, Mamede M, Preguiça N, Rodrigues R. Pixida: optimizing data parallel jobs in wide-area data analytics. Proc VLDB Endow. 2015;9(2):72–83.
23. Jonathan A, Chandra A, Weissman J. Awan: locality-aware resource manager for geo-distributed data-intensive applications. In: 2016 IEEE international conference on cloud engineering (IC2E); 2016. p. 32–41.
24. Zhao L, Yang Y, Munir A, Liu AX, Li Y, Qu W. Optimizing geo-distributed data analytics with coordinated task scheduling and routing. IEEE Trans Parallel Distrib Syst. 2020;31(2):279–93.
25. Huang Y, Shi Y, Zhong Z, Feng Y, Cheng J, Li J, et al. Yugong: geo-distributed data and job placement at scale. Proc VLDB Endow. 2019;12(12):2155–69.
26. Iordache A, Morin C, Parlavantzas N, Feller E, Riteau P, Resilin: elastic mapreduce over multiple clouds. In: 13th IEEE/ACM international symposium on cluster. Cloud, and Grid Computing. 2013;2013:261–8.
27. Ananthanarayanan R, Basker V, Das S, Gupta A, Jiang H, Qiu T, et al. Photon: fault-tolerant and scalable joining of continuous data streams. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data. SIGMOD '13. New York, NY, USA: Association for Computing Machinery; 2013. p. 577–88.
28. Hsieh K, Harlap A, Vijaykumar N, Konomis D, Ganger GR, Gibbons PB, et al. Gaia: geo-distributed machine learning approaching LAN speeds. In: 14th USENIX symposium on networked systems design and implementation (NSDI 17). Boston, MA: USENIX Association; 2017. p. 629–47.
29. Dolev S, Florissi P, Gudes E, Sharma S, Singer I. A survey on geographically distributed big-data processing using mapreduce. IEEE Trans Big Data. 2019;5(1):60–80.
30. Ji S, Li B. Wide area analytics for geographically distributed datacenters. Tsinghua Sci Technol. 2016;21(2):125–35.
31. WT Hadoop: The definitive guide. 4th ed. Newton: O'Reilly Media, Inc.; 2015.
32. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on networked systems design and implementation. NSDI'12. USENIX Association; 2012. p. 2.
33. Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I. Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles. SOSP '13. ACM; 2013. p. 423–38.
34. Carbone P, Katsifodimos A, Kth, Sweden S, Ewen S, Markl V, et al. Apache flink: stream and batch processing in a single engine. Bull IEEE Comput Soc Tech Committee Data Eng. 2015;38(4):28–38.
35. Apache Storm. http://storm.apache.org/.
36. European Commission press release. Commission to pursue role as honest broker in future global negotiations on internet governance. http://tinyurl.com/k8xcvy4.
37. Zhang X, Qian Z, Zhang S, Li Y, Li X, Wang X, et al. Towards reliable (and efficient) job executions in a practical geo-distributed data analytics system. arXiv e-prints. 2018, p. arXiv:1802.00245.
38. Viswanathan R, Ananthanarayanan G, Akella A. CLARINET: WAN-aware optimization for analytics queries. In: Proceedings of the 12th USENIX conference on operating systems design and implementation. OSDI'16. USENIX Association; 2016. p. 435–50.
39. Jain S, Kumar A, Mandal S, Ong J, Poutievski L, Singh A, et al. B4: experience with a globally-deployed software defined wan. SIGCOMM Comput Commun Rev. 2013;43(4):3–14.
40. Hong CY, Kandula S, Mahajan R, Zhang M, Gill V, Nanduri M, et al. Achieving high utilization with software-driven WAN. SIGCOMM Comput Commun Rev. 2013;43(4):15–26.
41. Calder M, Fan X, Hu Z, Katz-Bassett E, Heidemann J, Govindan R. Mapping the expansion of Google's serving infrastructure. In: Proceedings of the 2013 conference on internet measurement conference. IMC '13. ACM; 2013. p. 313–26.
42. Wang H, Li B. Lube: mitigating bottlenecks in wide area data analytics. In: Proceedings of the 9th USENIX conference on hot topics in cloud computing. HotCloud'17. USENIX Association; 2017. p. 1.
43. Costa PARS, Bai X, Ramos FMV, Correia M. Medusa: an efficient cloud fault-tolerant MapReduce. In: 201616th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid); 2016, p. 443–52.
44. Costa PARS, Ramos FMV, Correia M, Chrysaor: Fine-Grained, Fault-Tolerant Cloud-of-Clouds MapReduce. In: 2017 IEEE International Conference on Computer and Information Technology (CIT); 2017, p. 421–30.
45. Convolbo MW, Chou J, Hsu CH, Chung YC. GEODIS: towards the optimization of data locality-aware job scheduling in geo-distributed data centers. Computing. 2018;100(1):21–46.
46. Li P, Guo S, Miyazaki T, Liao X, Jin H, Zomaya AY, et al. Traffic-aware geo-distributed big data analytics with predictable job completion time. IEEE Trans Parallel Distrib Syst. 2017;28(6):1785–96.
47. Zhang G, Wang H, Luan Z, Wu W, Qian D. Improving performance for geo-distributed data process in wide-area. In: 2017 IEEE international conference on computer and information technology (CIT); 2017. p. 162–7.
48. Zhang H, Ramapantulu L, Teo YM. Harmony: an approach for geo-distributed processing of big-data applications. In: 2019 IEEE international conference on cluster computing (CLUSTER); 2019. p. 1–11.
49. Oh K, Chandra A, Network Weissman J. A system cost-aware geo-distributed data analytics. In: 20th IEEE/ACM international symposium on cluster. Cloud and Internet Computing (CCGRID). 2020, P. 649–58.
50. Wu D, Sakr S, Zhu L, Towards WuH, analytics big data, across multiple clusters. In: 17th IEEE/ACM international symposium on cluster. Cloud and Grid Computing (CCGRID). 2017,p. 218–27.
51. Wang H, Niu D, Li B. Dynamic and decentralized global analytics via machine learning. In: Proceedings of the ACM symposium on cloud computing. SoCC '18. ACM; 2018. p. 14–25.
52. Li H, Xu H, Nutanong S. Bohr: similarity aware geo-distributed data analytics. In: 9th USENIX workshop on hot topics in cloud computing (HotCloud 17). USENIX Association; 2017.
53. Li W, Niu D, Liu Y, Liu S, Li B. Wide-area spark streaming: automated routing and batch sizing. IEEE Trans Parallel Distrib Syst. 2019;30(6):1434–48.

54. Jonathan A, Chandra A, Weissman J. Multi-query optimization in wide-area streaming analytics. In: Proceedings of the ACM symposium on cloud computing. SoCC '18. ACM; 2018. p. 412–25.
55. Jonathan A, Chandra A, Weissman J. WASP: wide-area adaptive stream processing. In: Proceedings of the 21st international middleware conference. Middleware '20. New York, NY, USA: Association for Computing Machinery; 2020. p. 221–35.
56. Rabkin A, Arye M, Sen S, Pai VS, Freedman MJ. Aggregation and degradation in JetStream: streaming analytics in the wide area. In: Proceedings of the 11th USENIX conference on networked systems design and implementation. NSDI'14. USENIX Association; 2014. p. 275–88.
57. Costa PARS, Ramos FMV, Correia M. On the design of resilient multicloud mapreduce. IEEE Cloud Comput. 2017;4(4):74–82.
58. Chávez E, Navarro G, Baeza-Yates R, Marroquín JL. Searching in metric spaces. ACM Comput Surv. 2001;33(3):273–321.
59. Box GEP, Jenkins G. Time series analysis, forecasting and control. San Francisco: Holden-Day Inc.; 1990.
60. Chis T. Sliding hidden markov model for evaluating discrete data. In: Balsamo MS, Knottenbelt WJ, Marin A, editors. Computer performance engineering. Berlin Heidelberg: Springer; 2013. p. 251–62.
61. Ousterhout K, Wendell P, M, Stoica I. Sparrow: distributed, low latency scheduling. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles. SOSP '13. ACM; 2013. p. 69–84.
62. Gurobi Optimization. http://www.gurobi.com/.
63. Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proceedings of the 5th European conference on computer systems. EuroSys '10. ACM; 2010. p. 265–78.
64. Grandl R, Ananthanarayanan G, Kandula S, Rao S, Akella A. Multi-resource packing for cluster schedulers. SIGCOMM Comput Commun Rev. 2014;44(4):455–66.
65. TPC-DS Decision Support Benchmark. http://www.tpc.org/tpcds.
66. Big Data Benchmark. https://amplab.cs.berkeley.edu/benchmark/.
67. Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, et al. Apache Hadoop YARN: yet another resource negotiator. In: Proceedings of the 4th annual symposium on cloud computing. SOCC '13. ACM; 2013. p. 5:1–5:16.
68. Hunt P, Konar M, Junqueira FP, Reed B. ZooKeeper: wait-free coordination for internet-scale systems. In: Proceedings of the 2010 USENIX conference on USENIX annual technical conference. USENIXATC'10. USENIX Association; 2010. p. 11.
69. Cuzzocrea A, Bellatreche L, Song IY. Data Warehousing and OLAP over big data: current challenges and future research directions. In: Proceedings of the sixteenth international workshop on data warehousing and OLAP. DOLAP '13. ACM; 2013. p. 67–70.
70. Wang L, Tao J, Marten H, Streit A, Khan SU, Kolodziej J, et al. MapReduce across distributed clusters for data-intensive applications. In: 2012 IEEE 26th international parallel and distributed processing symposium workshops PhD Forum; 2012. p. 2004–11.
71. Conviva. https://www.conviva.com/datasheets/experience-benchmarks.
72. Boyd S, Parikh N, Chu E, Peleato B, Eckstein J. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and trends®in machine learning. 2011;3(1):1–122. 10.1561/2200000016.
73. Twitter Streaming API's. https://developer.twitter.com/en/docs.
74. Fan X, Lang B, Zhou Y, Zang T. Adding network bandwidth resource management to Hadoop YARN. In: 2017 seventh international conference on information science and technology (ICIST); 2017. p. 444–9.

Publisher's Note