MapReduce Frame Work: Investigating Suitability for Faster Data Analytics

Monali Mavani[1] and Leena Ragha[2]

[1] SIES College of Management Studies
Navi Mumbai, India
monamavani@gmail.com
[2] Ramrao Adik Institute of Technology
Navi Mumbai, India
leena.ragha@gmail.com

Abstract. Faster data analytics is the ability to generate the desired report in near real time. Any application that looks at an aggregated view of a stream of data can be considered as an analytic application. The demand to process vast amounts of data to produce various market trends, user behavior, fraud behavior etc. becomes not just useful, but critical to the success of the business. In the past few years, fast data, i.e., high-speed data streams, has also exploded in volume and availability. Prime examples include sensor data streams, real-time stock market data, and social-media feeds such as Twitter, Facebook etc. New models for distributed stream processing have been evolved over a time. This research investigates the suitability of Google's MapReduce (MR) parallel programming frame work for faster data processing. Originally MapReduce systems are geared towards batch processing. This paper proposes some optimizations to original MR framework for faster distributed data processing applications using distributed shared memory to store intermediate data and use of Remote Direct Access (RDMA) technology for faster data transfer across network.

Keywords: Map Reduce, faster data analytics, distributed shared memory, Remote Direct Memory Access.

1 Introduction

Today, with the growing use of mobile devices constantly connected to the Internet, the nature of User-generated data has changed, it has become more real-time. The New York Stock Exchange generates about one terabyte of new trade date per day, Facebook Hosts approximately 10 billion photos, taking up one petabyte of storage, The Internet Archives stores Around 2 petabyte of data, and is growing at a rate of 20 terabytes per month [1]. Processing data in batches is too slow to produce faster reports. Accumulated data can lose its importance in several hours or, even, minutes. Faster data processing requires stream processing and online aggregation. The process of mapping a request from the originator to the data source is called "Map"; and the

S. Unnikrishnan, S. Surve, and D. Bhoir (Eds.): ICAC3 2013, CCIS 361, pp. 119–130, 2013.

process of aggregating the results into a consolidated result is called "Reduce". This research investigates the suitability of Google's MapReduce (MR) parallel programming frame work for faster data processing. In the MR the user of the MR library expresses the computation as two functions: Map and Reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The Map Reduce library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. The Reduce function, also written by the user, accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically, just zero or one output value is produced per Reduce invocation. The key benefits of this model are that it harnesses compute and I/O parallelism on commodity hardware and can easily scale as the datasets grow in size [2]. Hadoop[3] is an open source MR framework used in distributed computing environment. This paper investigates the suitability of MR framework to be used in faster data analytics and takes Hadoop as a platform for study.

Apache is managing an open-source based Hadoop project, derived from the Nutch project [4]. The Hadoop project releases the Hadoop Distributed File System (HDFS) and Map Reduce framework. Following section explain the detail execution environment of Hadoop. Each map function is given a part of input data called input split which is coming from HDFS. The map function buffers the output in memory. Map output is intermediate output: it's processed by reduce tasks to produce the final output, and once the job is complete the map output can be thrown away. Before it writes to disk, the thread first divides the data into partitions corresponding to the reducers that they will ultimately be sent to. Within each partition, the background thread performs an in-memory sort by key. Hadoop allows the user to specify a combiner function to be run on the sorted map output—the combiner function's output forms the input to the reduce function. Running the combiner function makes for a more compact map output, so there is less data to write to local disk and to transfer to the reducer. Each time the memory buffer reaches the spill threshold, a new spill file is created, so after the map task has written its last output record there could be several spill files. Before the task is finished, the spill files are merged into a single partitioned and sorted output file. The map output is compressed as it is written to disk. The data flow between map and reduce tasks is colloquially known as "the shuffle," as each reduce task is fed by many map tasks. The output file's partitions are made available to the reducers over HTTP. So shuffle stage involves disk I/O and network I/O. Hadoop uses the concept of barrier to synchronize map and reduce operations. Map functions run in parallel, creating different intermediate values from different input data sets. All worker processes executing map functions reach to a barrier and then wait for reduce operation assignment to them by master process. So Bottleneck is that reduce phase cannot start until map phase completes. The output of the reduce is normally stored in HDFS for reliability. The map tasks may finish at different times, so the reduce task starts copying their outputs as soon as each completes. This is known as the copy phase of the reduce task. The map outputs are copied to the reduce task JVM's memory if they are small enough otherwise, they are copied to disk. When the in-memory buffer reaches a threshold size or reaches a threshold number of map outputs, it is merged and spilled to disk. When all the map outputs have been

copied, the reduce task moves into the sort phase which merges the map outputs, maintaining their sort ordering and applies reduce function to sorted data. It finally writes output to HDFS and informs the master about its location. In this execution environment heavy disk I/O and network I/O is observed.

Basic MR programming model gives simple yet powerful model for distributed data processing. MR model has got few limitations to be suitable for the execution environment for performing faster data analytics. Following section highlights its limitations.

The rest of the paper is organized as follows. Section 2 provides limitation of MR to be used directly in faster analytics. Section 3 presents literature survey. Section 4 proposes the optimization to MR frame work and Section 5 concludes the paper.

2 Faster Data Analytics

Faster data Analytics is the process of manipulating raw data and the ability to generate the desired report in near real time. Any application can be categorized that looks at an aggregated view of a stream of data, as an analytic application. Faster data Analytics involves stream processing and online aggregation. Many of the data intensive applications must combine new data with data derived from previous batches or iterate to produce results, and state is a fundamental requirement for doing so efficiently. For example, incremental analytics re-use prior computations, allowing outputs to be updated, not recomputed, when new data arrives [5]. To perform faster analytics, query results should be outputted as soon as input to the query is available rather than waiting for the whole input to get loaded. Reducers begin processing data as soon as it is produced by mappers, they can generate and refine an approximation of their final answer during the course of execution. This technique is known as online aggregation [6]. Faster data analytics requires fast in-memory processing of a MapReduce query program for all (or most) of the data. There are certain limitations of MR framework. Following subsection illustrate the same.

2.1 Limitations of Map Reduce for Faster Analytics

Various authors have identified limitation of MapReduce model for faster analytics following are some of the issues highlighted in various research.

- Group by implementation of SQL is implemented using sort-merge algorithm in MapReduce model which is CPU intensive.
- For huge datasets lot of intermediate data is generated so merge phase of sort merge step performs disk I/O due to memory overflow which is blocking.
- MapReduce systems wait until the entire data set is loaded to begin query processing and further employ batch processing of query programs, they are ill-suited for faster processing.
- Reduce phase cannot be started until map phase is over and all the data for the reduce function is available before starting the execution of reduce function. This further introduces latency.

- In shuffle phase a reduce task fetches data from each map task's local file system by issuing HTTP request. This involves network I/O as well as disk I/O which adds to latency.
- Traditional MapReduce implementations provide a poor interface for interactive data analysis, because they do not emit any output until the job has been executed to completion [7].
- Non suitability of MR for stream processing as mentioned by Lam et al. [8] is as follows. MapReduce runs on a static snapshot of a data set, while stream computations proceed over an evolving data stream. In MapReduce, the input data set does not (and cannot) change between the start of the computation and its finish. In stream computations, the data is changing all the time; there is no such thing as working with a "snapshot" of a stream.

Various researchers have worked on different solutions. They have addressed the possibility of faster and incremental data processing using MR framework in general and Hadoop in particular. Following section presents work done by different authors found in literature.

3 Literature Review

Faster data processing and online data aggregation is needed which becomes difficult by traditional MR framework due to above mentioned reasons. Various authors have suggested variant in MR framework to address above mentioned limitations.

- Condie et al. have proposed modified MR architecture -MapReduce Online, which implements Hadoop Online Prototype (HOP) with pipelining of data where map output is pushed eagerly to reducers [7]. Due to pipelining early returns on long-running jobs via online aggregation and continuous query over streaming data becomes possible. HOP pushes map output in finer granularity and hence increases network I/O cost and reduces CPU utilization. This moves some of the sorting work to reducers but still due to merge-sort implementation incurs disk I/O and n/w I/O.
- Elteir et al. have proposed enhancement in MR via asynchronous data processing namely incremental processing. They have tried to address drawback of synchronized communication between map and reduce of existing Hadoop implementations using two different approaches. The first approach, hierarchical reduction, starts a reduce task as soon as a predefined number of map tasks completes. The second approach, incremental reduction, starts a predefined number of reduce tasks from the beginning and has each reduce task incrementally reduce records collected from map tasks [9]. So they have tried to change run time system of original hadoop implementation for recursively reducible jobs. They evaluated different reducing approaches with two real applications on a 32-node cluster and found that incremental reduction can speed-up the original Hadoop implementation by up to 35.33% for the word count application and 57.98% for the grep application. But for hierarchical reduction network I/O cost will be high.
- Mazur et al. have addressed the drawback due to sort-merge phase of MR Runtime sequence which is CPU intensive affecting overall running time [2]. They have re-implemented sort –merge implementation of group by with hash based technique which can support fast in-memory processing whenever computation states of the

reduce function for all groups fit in memory. They have experimented the overhead due to sorting in the click stream analysis application and showed that up to 48% of CPU cycles, and up to 53% of running time can be saved. They also have experimented change at architecture level by using separate fast and small storage device to hold intermediate data for reducing disk contention and separate storage and computing nodes. Extra storage devices help reduces the total running time; from 76 minutes to 43 minutes for sessionization. Distributed storage system reduces the running time of sessionization from 76 minutes to 55 minutes. But the issues of blocking and intensive I/O remain unaffected.

- Bu et al. have proposed HaLoop, a runtime based on Hadoop which supports iterative data analysis applications especially for large-scale data [10]. By caching the related invariant data and reducers' local outputs for one job, it can execute recursively. It offers a programming interface to express iterative data analysis applications. HaLoop uses a new task scheduler for iterative applications that leverages data locality in the applications. From the original hadoop task scheduler and task tracker modules are modified, and the loop control, caching, and indexing modules are newly added modules. They evaluated HaLoop on real queries and real datasets. Compared with Hadoop, on average, HaLoop reduces query runtimes by 1.85, and shuffles only 4% of the data between mappers and reducers.
- Verma et al. [11] have presented techniques for supporting general purpose applications in a barrier-less MapReduce framework. They have modified Hadoop implementation. They have bypass the sorting mechanism, modified the invocation of the Reduce function so that it can be called with a single record. Reducer must maintain partial results for every key it has received. They used the Java implementation of Red-Black trees called TreeMap. They experimented seven different categories of MapReduce algorithms, and converted into barrier less version. Their results showed an average improvement of 25% (and 87% in the best case) in the job completion times, with minimal additional programmer effort. For large datasets huge intermediate results are generated which causes memory overflow and disk I/O. In order to address these memory overflow problems, they explored two possible memory management solutions: a disk spill and merge scheme and an off-the-shelf disk-spilling key-value store. In a disk spill and merge scheme when the memory usage reaches a predefined memory threshold, the structure moves the partial results to a newly created local spill file on the disk. Instead of flushing the entire contents of the memory to a file on the disk, the partial results can be maintained in a key/value store that has the capability of spilling to disk. They experimented BerkeleyDB (Java Edition) as key/value store. The disk spill and merge approach performed better than original hadoop. And disk-spillable key/value store has performed poorly on word count problem and they concluded that off-the-shelf key/-value store may not be suitable option for map reduce workload.
- Yan et al. have proposed IncMR framework which incrementally processing new data of a large data set, which takes state as implicit input and combines it with new data [12]. They have added additional modules and extended existing hadoop API. Map tasks are created according to new splits instead of entire splits while reduce tasks fetch their inputs including the state and the intermediate results of new map tasks from designate nodes or local nodes. Their work stresses on job scheduling based on data locality but lacks in state management issues like size and

location of state which incurs huge network I/O. In their work they highlighted the main overhead of IncMR in the storage and transmission of many intermediate results. Job scheduler will try to allocate reduce tasks to the nodes that have performed reduce tasks recently because they have cached related state.

- Lam et al. have describe Muppet - MapUpdate, a framework like MapReduce, but specifically developed for fast data. MapUpdate operates on data streams, so map and update functions must be defined with respect to streams [8]. Streams may never end, so updaters use storage called slates to summarize the data they have seen so far. In MapUpdate, slates are "memories" of updaters, distributed across multiple map/update machines as well as persisted in a key value store for later processing. It is continuously updated an in-memory data structure that stores all important information that the update function U must keep about all the events with key k that U has seen so far. Muppet uses Cassandra key-value store to store slates. They have run Cassandra key-value store on solid-state flash-memory storage (SSDs). In further optimization the workers pass events directly to one another without going through any master. The master in Muppet is used for handling failures. Muppet uses hashing techniques in shuffle stage.
- Yocum et al. have implemented MapReduce over a distributed stream processor. They have used in-network aggregate computation that avoids data reprocessing. They have used window based stream processing input is in streams rather than batches. Their system shows efficiency for huge unstructured data. They have used the distributed stream processing platform, Mortar to create and manage the physical MapReduce data flows. Mortar is a platform for instrumenting end hosts with user-defined stream processing operators. The platform manages the creation and removal of operators, and orchestrates the flow of data between them.
- Logothetis et al. have proposed a generalized architecture for continuous bulk processing (CBP) where prior results are reused to incrementally incorporate the changes in the input [5]. They have proposed group wise processing operator, translate, and dataflow primitives to maintain state during continuous bulk data processing. The translate operator is run repeatedly, allowing users to easily store and retrieve state as new data inputs arrive. They have modified hadoop to support full CBP model and optimize treatment of state. They have evaluated their CBP model on incremental crawl queue, clustering coefficients and page rank. In Page Rank application data movement is reduced by46 % and running time is cut by 53% reducing both network and CPU usage.
- Bhatotia et al. have proposed the architecture- Incoop based on incremental HDFS (content based input splitting), incremental map (memorization aware scheduler), incremental reduce (combiners for each reduce task) to maximize the reuse of results of previous computations [22]. Work and time speedups vary between 3-fold and 1000-fold for incremental modifications ranging from 0% to 25% of data. Higher speedups are observed for computation-intensive applications (K-Means, KNN) than for data-intensive applications (Word Count, Co-Matrix, and BiCount). Both work and time speedups decrease as the size of the incremental change increases, because larger changes allow fewer computation results from previous runs to be reused. Their evaluation shows that Incoop can improve efficiency in incremental runs, at a modest cost in the initial, first run where no computations can be reused.

Various methods have been proposed for faster data processing which involves either modifying the Hadoop API or extending the API by adding certain modules. Some of them have proposed changes in the runtime system or suggested architectural level changes. But disk I/O and network I/O still remains as the bottleneck. This paper proposes optimizations in original framework which can help in reducing latency due to disk I/O and network I/O.

4 Proposed Optimization

This research tries to address some of the limitations of MR model for fast data analytics. It proposes

- Use of distributed shared memory (shared tuple space) to store intermediate results.
- Use of Remote Direct Memory Access (RDMA) to address the latencies introduced due to MR execution environment.

This work is also in favor of using stateful map reduce framework. Figure 1 shows Architecture based on RDMA and tuple space.

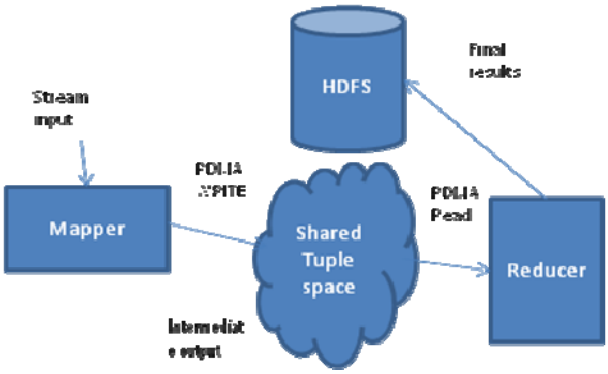

Fig. 1. Architecture based on RDMA and tuple space

1. Use of stateful approach during the execution of one MR Job helps in reduce the latency in producing results. Use of stateful approach in consecutive MR jobs helps iterative algorithms to be processed faster and more efficient way with MR framework. Stateful reducer allows partial results to be saved as state which can be reused with the new input and emitted as intermediate results for interactive query processing. In case of stream processing input continuously arrives. Mapper can store partially produce results (key, value pair) as a state and with each new input from the stream updates the state with corresponding key.
2. Mappers output can be written in shared tuple space. Reducer will start fetching data from shared tuple space and starts early execution. Inputs are fetched by reducer as it is emitted by mapper, so periodically slot in tuple space becomes available which can be reused to store another map output. Map outputs are deleted in shared space as soon as it is taken by reducer, so to have fault tolerance; map

outputs are also written to local disk of mapper's nodes. This induces disk I/O but does not affect execution time. A tuple space is an implementation of the associative memory paradigm for parallel/distributed computing. Producers post their data as tuples in the space, and the consumers then retrieve data from the space that match a certain pattern.[14] Linda [15] language was the first implementation of Tuple Spaces. Later on, the JavaSpaces specification was defined as part of Sun's Jini specification. The JavaSpaces technology [16] is a simple and powerful high-level tool for building distributed and collaborative applications based on the concept of shared network-based space that serves as both object storage and exchange area. The space has a built-in clustering model, which allows it to share data and messages with other spaces on the network. [17]

3. Network I/O latency can be reduced with use of Remote Direct Memory Access (RDMA), next generation network stack for high speed-high performance computing. Modern RDMA capable networks such as InfiniBand and Quadrics provide low latency of a few microseconds and high bandwidth of up to 10 Gbps. This has significantly reduced the latency gap between access to local memory and remote memory in modern clusters. [18]. RDMA provides read and write services directly to applications and enables data to be transferred directly into Upper Layer Protocol (ULP). It also enables a kernel bypass implementation. [19] InfiniBand [20] is an I/O architecture designed to increase the communication speed between CPUs and subsystems located throughout a network. One of the most important features of IB is RDMA. It provides applications with an easy-to-use messaging service. Instead of making a request to the operating system for access to one of the server's communication resources, an application accesses the InfiniBand messaging service directly. InfiniBand provides two transfer semantics; a channel semantic sometimes called SEND/RECEIVE and a pair of memory semantics called RDMA READ and RDMA WRITE. The "initiator" (mapper) places the block of data in a buffer in its virtual address space and uses a SEND operation to send a storage request to the "target" (e.g. the storage device –shared tuple space). The target, in turn, uses RDMA READ operations to fetch the block of data from the initiator's (mappers) virtual buffer. Once it has completed the operation, the target uses a SEND operation to return ending status to the initiator. The initiator (mapper), having requested service from the target, was free to go about its other business while the target asynchronouslv completed the storage operation, notifying the initiator (mapper) on completion. Similarly when reducer reads the data from shared space it uses receive operation to send a fetch request to the target. The target, in turn, uses RDMA write operation to write the data to reducer's internal virtual buffer. Once it has completed the operation, the target uses a SEND operation to return ending status to the reducer. The reducer having requested service from the target (shared space), was free to execute its code while the target asynchronously completed the storage operation, notifying the reducer on completion. This prefetch of input overlapped with reducer execution improves the performance and reduces the latency. The Sockets Direct Protocol (SDP) is a networking protocol developed to support stream connections over InfiniBand fabric. It acts as an interface between high-level application and IB fabric [21].

So with the use of RDMA aware communication channel and distributed shared memory to store intermediate data delay in execution can be reduced. These are few optimizations suggested to improve MR model in order to be suitable execution environment for faster data analytics. In this research we have tried to verify our proposal using performance model developed by Krevat et al. Next section describes performance modeling in terms of overall execution speed.

5 Performance Modeling

This section verifies the suggested optimization in improving speed of data analytics operation using distributed shared memory unit tuple space. For the reference performance model developed at Parallel Data Laboratory, Carnegie Mellon University is taken [23].

5.1 Assumptions

- data-intensive workloads
- computation time is negligible in comparison to I/O speeds
- assumption holds for grep- and sort-like jobs
- pipelined parallelism can allow non-I/O operations to execute entirely in parallel with I/O operations
- input data is evenly distributed across all participating nodes in the cluster
- nodes are homogeneous
- each node retrieves its initial input from local storage
- a single job has full access to the cluster at a time,with no competing jobs or other activities

Table 1 identifies the I/O operations in each map-reduce phase for two variants of the sort operator.

Table 1. Configuration parameters for performance modeling

Symbol	Definition
n	no.of nodes in the cluster
Dw	The aggregate d disk write throughput of a single node
Dr	The aggregate d disk read throughput of a single node
eM	The ratio between Map operator's output and input
eR	The ratio between reduce operator's output and input
N	The network throughput of a node
r	Replication factor used for the job's output data if no replication then r=1
I	Total amount of I/p data for a given computation

When intermediate data does not fit in memory then in traditional map reduce operation, an external sort is used where data is written on disk in multiple files and sorting phase is applied i.e. in disk sorting is used. Overall execution time for external sort bearing above assumptions can be calculated by following equation derived and verified by Krevat et al.:

$$i/n*\{\max[(1\div Dr)+(Em\div Dw),(n-1)*Em\div n*N]+\max[(Em\div Dr)+r*Em*Er\div Dw, Em*Er*(r-1)\div N])\} \tag{1}$$

Overall execution time is coming out to be 462.0288 seconds for 1TB of i/p data, 25 node clusters, disk throughput is 780 MB/s, Network throughput is 110MB/s and replication factor for output is 1.

Now with the use of tuple space even with larger intermediate data that does not fit into memory of the node, external sort can be avoided and in memory sorting can be used where now memory is in the distributed form. Following equation for in memory sorting is used:

$$i/n*\{\max(1\div Dr,(n-1)\div n)Em\div N)+\max(r*Em*Er\div Dw,[Em*Er*(r-1)\div N])\} \tag{2}$$

Overall execution time is reduced to 409.6 seconds for the same configuration. This shows improvement due to bypassing disk operations.

Authors of this model have verified their developed model using Parallel Data Series (PDA) -data analysis tool. They have also used this model for modeling various published benchmarks. Then they have demonstrated inefficiency of Hadoop for optimal configurations. Motivation to use this model to verify our suggested optimization is that this model accurately takes different system parameters in to account. As second equation calculates total execution time when disk operations are not needed and they have used this when intermediate data are fitted in single node's memory . However even if data is more than the capacity of single node's memory, it can be distributed across different node's memory and can be managed by layer above it using tuple space API. So second equation still holds true for distributed shared memory.

With the use of tuplespace disk I/O can be avoided but network I/O still remains as the speed limiting factor. As proposed in the previous section next generation interconnection technology based on Remote Direct Memory Access can reduce latency by bypassing Tcp stack which is totally controlled by kernel for network data transfer.

6 Conclusion

This paper investigates the suitability of MR frame work to use in faster data analytics and suggests optimization in traditional MR framework. Use of stateful execution engine helps mappers and reducers to execute faster by updating the state rather than generating the output as inputs come in. Asynchronous communication between mapper and reducer helps to start early execution of reduce function which lowers overall the execution time and less intermediate data to be handled and stored by MR runtime

engine. Due to this, shared tuple space, a distributed shared memory unit, can be used to store intermediate data instead of mapper's local disk. Further optimization can be achieved through the use of Remote Direct Memory Access (RDMA) to reduce latency due to network I/O. As a part of future work, suggested optimizations can be evaluated in real experimental setup.

References

1. White, T.: Hadoop The Definitive Guide. Yahoo Press (January 2012)
2. Li, B., Mazur, E., Diao, Y., McGregor, A., Shenoy, P.: A Platform for Scalable One-Pass Analytics using MapReduce. In: International Conference on Management of Data, pp. 985–996 (2011)
3. http://hadoop.apache.org
4. Khare, R., Sitaker, D.C.K., Rifkin, A.: Nutch: A Flexible and Scalable Open-Source Web Search Engine, Oregon State University, Commerce Net Labs Technical Report, pp. 1–10 (2004)
5. Logothetis, D., Olston, C., Reed, B.: Stateful Bulk Processing for Incremental Analytics. In: Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 51–62 (2010)
6. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online Aggregation. In: SIGMOD Conference, pp. 171–182 (1997)
7. Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Elmeleegy, K., Sears, R.: Map reduce online. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, pp. 1–14 (2010)
8. Lam, W., Liu, L., Prasad, S.T.S., Rajaraman, A., Vacheri, Z., Doan, A.H.: Muppet: MapReduceStyle Processing of Fast Data. Proceedings of the VLDB Endowment 5(12), 1814–1825 (2012)
9. Elteir, M., Lin, H., Feng, W.-C.: Enhancing mapreduce via asynchronous data processing. In: IEEE 16th International Conference on Parallel and Distributed Systems, pp. 397–405 (2010)
10. Bu, Y., Howe, B., Balazinska, M., Ernst, M.: Haloop: efficient iterative data processing on large clusters. In: 34th International Conference on Very Large Data Bases, pp. 285–296 (2010)
11. Verma, A., Zea, N., Cho, B., Gupta, I., Campbell, R.H.: Breaking the MapReduce Stage Barrier. Journal on Cluster Computing, 1–16 (2011)
12. Yan, C., Yang, X., Yu, Z., Li, M., Li, X.: IncMR: Incremental Data Processing based on MapReduce. In: 5th IEEE International Conference on Cloud Computing, pp. 534–541 (2012)
13. Logothetis, D., Yocum, K.: AdHoc Data Processing in the Cloud. Proceedings of the VLDB Endowment 1(2), 1472–1475 (2008)
14. http://en.wikipedia.org/wiki/Tuple_space
15. http://en.wikipedia.org/wiki/Linda_coordination_language
16. http://java.sun.com/developer/technicalArticles/tools/JavaSpaces
17. http://wiki.gigaspaces.com/wiki
18. Liang, H., Noronha, R., Panda, D.K.: Swapping to Remote Memory over InfiniBand: An S Approach using a High Performance Network Block Device. In: IEEE International Conference on Cluster Computing, pp. 1–10 (2005)

19. Recio, R., Metzler, B., Culley, P., Hilland, J., Garcia, D.: A Remote Direct Memory Access Protocol Specification. RFC 5040
20. http://members.infinibandta.org/kwspub/Intro_to_IB_for_End_Users.pdf
21. http://docs.oracle.com/javase/tutorial/sdp
22. Bhatotia, P., Wieder, A., Rodrigues, R., Acar, U.A., Pasquini, R.: Incoop: MapReduce for Incremental Computations. In: Proceedings of the 2nd ACM Symposium on Cloud Computing, Article No. 7, pp. 1–14 (2011)
23. Krevat, E., Shiran, T., Anderson, E., Tucek, J., Wylie, J.J., Ganger, G.R.: Applying performance models to understand data-intensive computing efficiency. Technical Report Carnegie Mellon University, HP Labs, pp. 1–18 (2010)