International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE)
ISSN: 0976-1353 Volume 24 Issue 3 – FEBRUARY 2017 (SPECIAL ISSUE).

A SURVEY OF STREAM PROCESSING ON HETEROGENEOUS CLUSTERS

A.Vignesh [#1], K.Dhakshnamurthy [*2] and D.B.Shanmugam [*3]

[#1] *M.Phil, Research Scholar, Dr.M.G.R.Chockalingam Arts College, Arni.*

[*2] *Assistant Professor, Department of BCA, King Nandhivarman College of Arts & science, Thellar*

[*3] *Associate Professor, Department of MCA, Sri Balaji Chockalingam Engineering College , Arni*

***Abstract*— Big data processing is a hot topic of today's computer world. One of the key paradigms behind it is MapReduce—parallel and massively distributed model inspired by the map and reduce functions commonly used in functional programming. Due to its simplicity and general availability of standard implementations, the paradigm has been massively adopted on current computer clusters. Yet, MapReduce is not optimal for all big data problems. My work focuses on the area of an alternative paradigm—stream processing—which has multiple advantages over the MapReduce, e.g., it avoids persistent data storing if not required. The research aims at overcoming deficiencies of existing stream processing frameworks that prevent its wider adoption.**

Basic scheduling decisions are discussed and demonstrated on naive scheduling of a sample application. The paper presents a proposal of a novel scheduler for stream processing frameworks on heterogeneous clusters, which employs design-time knowledge as well as benchmarking techniques to achieve optimal resource-aware deployment of applications over the clusters and eventually better overall utilization of the cluster.

In particular, the work deals with scheduling problems of stream processing applications on heterogeneous clusters. Heterogeneity is a typical characteristic of today's large data centers (caused by incremental upgrades and combinations of computing architectures, including specialized hardware such as GPU or FPGA) and advanced scheduling mechanisms can significantly increase efficiency of their utilization. The state-of-the-art research and development of stream processing and advanced methods of related scheduling techniques are discussed in this document. A special attention is paid to benchmark-based scheduling for distributed stream processing which also forms the core of my previous work and the proposed research towards my doctoral thesis. Finally, the concept of novel heterogeneity aware scheduler is presented first in the intuitive way and then discussed deeper on theoretical basis. The prototype of the scheduler is then described and promising results of basic experiments are showed.

***Index Terms*—heterogeneous clusters, survey, map reduce, big data**

I. INTRODUCTION

In homogeneous computing environments, all nodes have identical performance and capacity. Resources can be allocated evenly across all available nodes and effective task scheduling is determined by quantity of the nodes, not by their individual quality. Typically, resource allocation and scheduling in the homogeneous computing environments balance of workload across all the nodes which should have identical workload. Contrary to the homogeneous computing environments, there are different types of nodes in a heterogeneous cluster with various computing performance and capacity. High-performance nodes can complete the processing of identical data faster than low-performance nodes. Moreover, the performance of the nodes depends on the character of computation and on the character of input data. For example, graphic-intensive computations will run faster on nodes that are equipped with powerful GPUs while memory-intensive computation will run faster on nodes with large amount of RAM or disk space.

To balance workload in a heterogeneous cluster optimally, a scheduler has to (1) know performance characteristics for individual types of nodes employed in the cluster for different types of computations and to (2) know or to be able to analyse computation characteristics of incoming tasks and input data. The first requirement, i.e., the performance characteristics for individual types of employed nodes, means the awareness of infrastructure and topology of a cluster including detailed specification of its individual nodes. In the most cases, this information is provided at the cluster design-time by its administrators and architects. Moreover, the performance characteristics for individual nodes employed in a cluster can be adjusted at the cluster's run-time based on historical data of performance monitoring and their statistical analysis of processing different types of computations and data by different types of nodes. The second requirement is the knowledge or the ability to analyse computation characteristics of incoming tasks and input data. In batch processing, tasks and data in a batch can be annotated or analysed in advance, i.e., before the batch is executed, and acquired knowledge can be utilized in optimal allocation of resources and efficient task scheduling. In stream processing, the second requirement is much more difficult to meet due to continuous flow and unpredictable variability of the input data which make thorough analysis of computation characteristics of the input data and incoming tasks impossible, especially with real-time limitations in their processing.

To address the above mentioned issues of stream processing in heterogeneous clusters with optimal performance, user defined tasks processing (at least some) of the input data has to help the scheduler. For example, an application may

include user-defined helper-tasks tagging input data at run-time by their expected computation characteristics for better scheduling1. Moreover, individual tasks of a stream application should be tagged at design-time according to their required computation resources and real-time constraints on the processing to help with their future scheduling. Implementation of the mentioned tagging of tasks at design-time should be part of modeling (a meta-model) of topology and infrastructure of such applications.

II. Typical Uses of Heterogeneous Networks

In this section, we outline how heterogeneous networks of computers are typically used by their end-users. In general, heterogeneous networks are used traditionally, for parallel computing or for distributed computing. The traditional use means that the network of computers is used just as an extension of the user's computer. This computer can be serial or parallel. The application to be run is a traditional application, that is, the one that can be executed on the user's computer. The code of the application and input data are provided by the user. The only difference from the fully traditional execution of the application is that it can be executed not only on the user's computer but on any other relevant computer of the network. The decision where to execute one or other application is made by the operating environment and mainly aimed at better utilization of available computing resources (e.g., at higher throughput of the network of computers as a whole multi-user computer system). Faster execution of each individual application is not the main goal of the operating environment but can be achieved for some applications as a side-effect of its scheduling policy. This use of the heterogeneous network assumes the application and hence the software is portable and can be run on another computing resource. This assumption may not be true for some applications.

A heterogeneous network of computers can be used for parallel computing. The network is used as a parallel computer system in order to accelerate the solution of a single problem. In this case, the user provides a dedicated parallel application written to efficiently solve the problem on the heterogeneous network of computers. High performance is the main goal of that type of use. As in the case of traditional use, the user provides both the (source) code of the application and input data. In the general case, when all computers of the network are of the different architecture, the source code is sent to the computers where it is locally compiled. All the computers are supposed to provide all libraries necessary to produce local executables.

A heterogeneous network of computers can be also used for distributed computing. In the case of parallel computing, the application can be executed on the user's computer or on any other single computer of the network. The only reason to involve more than one computer is to accelerate the execution of the application. Unlike parallel computing, distributed computing deals with situations when the application cannot be executed on the user's computer just because not all components of the application are available on this computer. One such a situation is that some components of the code of the application cannot be provided by the user and are only available on remote computers. The reasons behind this can be various: the user's computer may not have the resources to execute such a code component, or the efforts and amount of resources needed to install the code component on the user's computer are too significant compared with the frequency of its execution, or this code may be just not available for installation, or it may make sense to execute this code only on the remote processor (say, associated with an ATM machine), etc.

Another situation is that some components of input data for this application cannot be provided by the user and reside on remote storage devices. For example, the size of the data may be too big for the disk storage of the user's computer, or the data for the application are provided by some external party (remote scientific device, remote data base, remote application, and so on), or the executable file may not be compatible with the machine architecture, etc.

The most complex is the situation when both some components of the code of the application and some components of its input data are not available on the user's computer. Performance. An immediate implication from the heterogeneity of processors in a network of computers is that the processors run at different speeds. A1 good parallel application for a homogeneous distributed memory multiprocessor system tries to evenly distribute computations over available processors. This very distribution ensures the maximal speedup on the system consisting of identical processors. If the processors run at different speeds, faster processors will quickly perform their part of computations and begin waiting for slower ones at points of synchronization and data transfer. Therefore, the total time of computations will be determined by the time elapsed on the slowest processor. In other words, when executing parallel applications, which evenly distribute computations among available processors, the heterogeneous network will demonstrate the same performance as a network of interconnected identical processors equivalent the slowest processor of the heterogeneous network of computers.

Therefore, a good parallel application for the heterogeneous network must distribute computations unevenly taking into account the difference in processor speed. The faster the processor is, the more computations it must perform. Ideally, the volume of computation performed by a processor should be proportional to its speed. Let us summarize our brief analysis of the performance-related programming issues in parallel and distributed computing for heterogeneous networks of computers.

There are two basic related issues that should be addressed:

• Performance model of heterogeneous network of computers quantifying the ability of the network to perform computations and communications,

• Performance model of application quantifying the computations and communications to be performed by the application.

In the presence of such models, given parameters of the

application model, parameters of the model of the executing network of computers and a mapping of the application to the processors of the network, it would be possible to calculate the execution time of the application on the network. Application and system programmers could incorporate the models in their applications and programming systems to achieve better distribution of computations and communications leading to higher performance. The performance model of heterogeneous network of computers includes two submodels:

• Performance model of a set of heterogeneous processors which is used to estimate the execution time of computations,

• Performance model of communication network which is necessary to predict the execution time of communication operations.

Now we briefly outline performance models currently used in programming systems and operating environments for heterogeneous networks of computers.

III. Current Development

Recently, based on the popularity of MapReduce and the wide spread of Hadoop, there were introduced series of systems exploiting ideas of a MapReduce paradigm in context of the stream processing. As was already mentioned in the Introduction, the developers of the Hadoop Online extended the original Hadoop system by ability to stream intermediate results from the map to the reduce tasks as well as the possibility to pipeline data across the different MapReduce jobs.

To facilitate these new features, they extended the semantics of the classic reduce function by time-based sliding windows, picked up this idea and further improved the suitability of Hadoop-based systems for continuous streams by replacing the sort-merge implementation for partitioning by a new hash-based approach.

The Muppet system also focuses on the parallel processing of continuous stream data while preserving a MapReduce-like programming abstraction. However, the authors decided to replace the reduce function by a more generic update function to allow for greater flexibility when processing intermediate data with identical keys. Muppet also aims to support near-real-time processing latencies.

The systems and Storm can also be classified as massively-parallel data processing systems with a clear emphasis on low latency. Their programming abstraction is finally not MapReduce but allows the developers to assemble arbitrarily complex DAG of processing tasks. For example, Twitter Storm does not use the intermediate queues to pass the data items from one task to the other; instead, data items are passed directly between the tasks using batch messages on the network level to achieve a good balance between latency and throughput.

In the end, it is important to note that along with the stream processing paradigm we can lately observe another movement in the field of low latency computations based on the fast message brokers such as Apache Kafka or Apache Flume. Although the message brokers are well known concept, the new wave of this technology focuses on different objectives, which makes it more suitable for high throughput processing. Traditional enterprise messaging systems e.g., IBM Websphere MQ, or JMS specification compliant brokers, emphasize strong delivery guarantees mostly with pushing data to consumers and ignore the throughput, which makes them very robust and often slow. The new wave of message delivery systems, on the other hand, accentuates throughput and lets consumers to pull data as they need. This opens a way for the new distributed applications with high throughput and straightforward design constructed on top of these queues. In my opinion, such message delivery systems are another kind of stream processing systems with less limited communication schemas i.e., DAGs are welcomed but not required. This was a brief overview of "historical" and recent approaches to the stream processing. Because this work deals mainly with the resource allocation and scheduling, from now on we will discuss the parallel systems only.

IV. Components and Architecture

In this section, the focus will be given to S4 and Storm because the overview of wider range of systems would be excessively long, and, at the same time, we can consider these two being representative examples of modern stream processing architectures. Across many massively parallel systems, a kind of a master-worker pattern is very common. The master node usually receives data or tasks and distributes them over the network of worker nodes. A good example could be again the MapReduce—its master process after receiving a job descriptor starts mappers and reducers on different machines; at the same time, the master process is responsible for a fault tolerance and liveness of the worker nodes (see Fig. 1). For parallel stream systems where the streams of data are often running for a long time, the placement of tasks can be less frequent but the concept of master and worker nodes stays unchanged. One interesting difference is that MapReduce job eventually finishes, whereas stream processing topologies run forever.

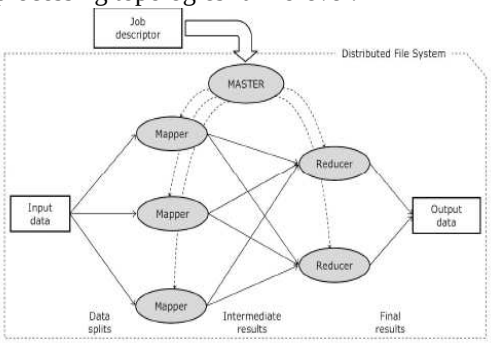

Fig. 1 Master and worker nodes (mappers and reducers) in MapReduce, reproduced form

When we look deeper into the structure of parallel systems, we can see that the computations are divided into jobs. The arrangement of jobs can be described by a graph with vertices representing the job's individual tasks and the edges denoting communication channels between them. For example, from a high-level perspective, the graph representation of a typical MapReduce job would consist of a set of Map vertices connected to a set of Reduce vertices.

Some frameworks have generalized the MapReduce model to arbitrary directed acyclic graphs (DAGs), some even allow graph structures containing loops. The stream processing

systems are very similar to other parallel systems. The computation is mostly described as a DAG. There are slight differences in a communication layer, while some systems use queues and intermediate message brokers; the others use a straight point to point (process to process) communication. Let me now describe the example topologies of S4 and Storm. Heterogeneity of large clusters and cloud setups is inevitable over the time. Main reasons are the need for growth in time and gradual upgrades of hardware. Simultaneously, the right treatment of computing resources with different capabilities can potentially bring better performance in the means of increased number of completed jobs per time period and thus the shorter time from queued to finished of each job.

V. Conclusion

This paper overviews my current knowledge and understanding of the topic of stream processing systems' scheduling on heterogeneous clusters and proposes a plan for the future research work towards the doctoral thesis. The introduction section has history and basic notions of stream processing, scheduling, and benchmarking. Argumentation about the present unsatisfying situation of stream processing regarding the integration into the big data ecosystems, e.g., Hadoop, about the lack of advanced scheduling algorithms, and about the problems accompanying heterogeneous clusters is discussed. Separated sections look into the solutions exploiting the benchmark-based scheduling in combination with other advanced scheduling techniques and to description of areas that currently lack deeper research. Besides that, my previous work and affiliation lying in multicloud middleware mOSAIC and in processing of scientific papers in grid and cloud (ReReSearch) was presented. Main part of the work then brings the motivations for my research with intuitive example of potentials laying in exploitation of heterogeneity of clusters. Basic ideas and dependencies of benchmark based scheduling are then shown and connected.

REFERENCES

[1] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy and R. Sears, "MapReduce Online," 2010.
[2] E. Mazur, B. Li, Y. Diao and P. Shenoy, "Towards Scalable One-Pass Analytics Using MapReduce," pp. 1102--1111, 2011.
[3] B. Lohrmann, D. Warneke and O. Kao, "Nephele streaming: stream processing under QoS constraints at scale," Cluster Computing, pp. 1-18, 2013.
[4] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker and N. Tatbul, "Aurora: a new model and architecture for data stream management," The Vldb Journal, vol. 12, no. 2, pp. 120--139, 2003.
[5] S. Babu and J. Widom, "Continuous queries over data streams," Sigmod Record, vol. 30, no. 3, pp. 109--120, 2001.
[6] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein and R. Varma, "Query Processing, Approximation, and Resource Management in a Data Stream Management System," 2003.
[7] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel and Y. Xing, "Scalable Distributed Stream Processing," 2003.
[8] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-h. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul and Y. Xing, "The Design of the Borealis Stream Processing Engine," 2005.
[9] M. Isard, M. Budiu, Y. Yu, A. Birrell and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," 2007.