HPC2-ARS: An Architecture for Real-time Analysis of Big Data Streams

Conference Paper · July 2018

DOI: 10.1109/ICWS.2018.00051

4 authors:

Yingchao Cheng
GuangDong University of Technology
4 PUBLICATIONS **1** CITATION
SEE PROFILE

Zhifeng Hao
GuangDong University of Technology
7 PUBLICATIONS **99** CITATIONS
SEE PROFILE

Ruichu Cai
64 PUBLICATIONS **440** CITATIONS
SEE PROFILE

Wen Wen
GuangDong University of Technology
27 PUBLICATIONS **155** CITATIONS
SEE PROFILE

HPC2-ARS: an Architecture for Real-time Analytic of Big Data Streams

Yingchao Cheng, Ruichu Cai, Wen Wen
School of Computer Science and Technology
Guangdong University of Technology
Guangzhou, China
chengyingchao@tamu.edu

Zhifeng Hao
School of Mathematics and Big Data
Foshan University
Foshan, China
zfhao@fosu.edu.cn

***Abstract*—HPC2-ARS supports a high performance cloud computing (HPC2) based streaming data analytic system, which ensures real-time response on unpredictable and fluctuating Big Data Streams by provisioning and scheduling computing resources autonomously. It focuses on parallel high-volume streaming applications, which have stringent real-time constraints and bring Big Data issues. It is a brand-new three-layered architecture, which solves three essential problems: (a) how many resources are needed for each application to achieve real-time analytic on streaming Big Data, (b) where to best place the allocated resources to minimize resource consumption, and (c) how to minimize response time for parallel applications. In summary, HPC2-ARS provides high performance streaming services.**

Keywords-Big Data Streams; cloud computing; real-time analytic; autonomous resource scheduling

I. INTRODUCTION

Information is increasing at exponential rate simultaneously, while the improvement of information analytic methods is relatively slower [1]. What's more, no one size fits all tactic for Big Data. As the information keeps increasing, today's Big Data problem will surely become the small data set problem in the future [2]. Thus, it is always a huge challenge to deal with Big Data issues.

In many high-volume, growing, and autonomous data sources (such as web click streams), data is in motion with high velocity, which is known as Big Data Streams. The common solutions for stream processing is to place a streaming data processing system on top of a cloud infrastructure [3]. However, when it comes to Big Data Streams analytic, the traditional streaming data processing systems are facing tremendous challenges. The data repositories may exceed Exabyte, increasing rapidly. Beyond their sheer magnitude, Big Data Streams and associated applications' considerations pose significant challenges for method and software development [4]. Big Data Streams require continuous queries over high-volumes unbounded data. To overcome it, both characteristics of hardware platform and the software stack do fundamentally matter. Analyzing Big Data in a timely manner necessitates a high performance and scalable computing infrastructure [5].

Motivated by this, we propose a three-layered architecture, HPC2-ARS shown in Fig. 1, which provides a HPC2-based streaming data analytic system that is capable to significantly satisfy the requirements of real-time Big Data Streams analytic, by offering flexible and scalable HPC computing resources autonomously.

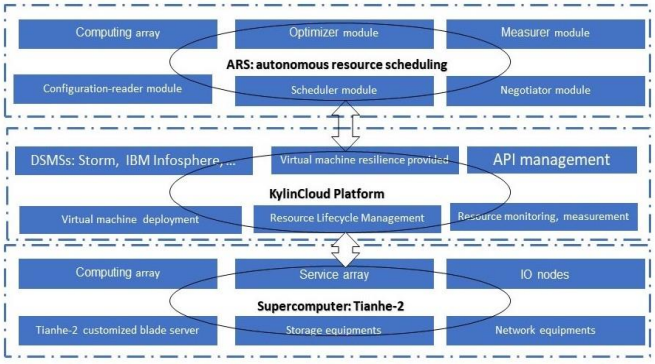

Fig. 1. The Illustration of HPC2-ARS implementation.

Even though the statistical characteristics of Big Data Streams are unpredictably fluctuating [6], HPC2-ARS is able to achieve two goals by autonomously provisioning resources: (1) ensure real-time response with minimum resource consumption for each streaming application, (2) speed up real-time analytic of the streaming data processing systems as fast as possible. HPC2-ARS is a suitable architecture for elastic virtual resources management, providing the capacity and environment that exactly matches real-time demand of Big Data Streams analytic. It is also a general systems approach over varying stream spikes, heterogeneous compute requirements, etc. In particular, we take Tianhe-2 supercomputer [7] as the first layer of HPC2-ARS, providing huge computing power (up to 33.86 Pflops). Its second layer is Kylin Cloud platform (custom OpenStack), providing virtually unlimited computing resources. The third layer is an autonomous resource-scheduling module that is on the top of the Kylin Cloud platform. HPC2-ARS combines the advantages of HPC, cloud computing, and streaming services, providing flexible tremendous resources for real-time analytic on Big Data Streams.

II. RELATED WORK AND ARGUMENTATION

High performance computing has appeared because of increasing demand for Big Data processing. It allows the use of supercomputers and parallel processing, in order to solve complex problems that would otherwise require unreasonable amounts of time on ordinary machines. Meanwhile, cloud computing is successfully used to harness

the power of large-scale distributed systems. In addition, it has been revolutionizing the IT industry by adding flexibility to the way IT is consumed, enabling infrastructures to be scaled up and down rapidly, adapting the systems to the actual demand [8]. When HPC and cloud computing merge, they give birth to HPC2, which essentially means applying cloud based environments to supercomputers. HPC2 promises instantaneous access to computational resources at arbitrary scale, making HPC systems more flexible and cost efficient [9]. Therefore, we introduce HPC2 to deal with real-time Big Data Streams analytic, and develop an autonomous resource-scheduling module to optimize this procedure.

We are facing one of the key challenges of HPC2, which is performance modeling of online processing and autonomous resource scheduling [10]. There are mature works [11], [12], [13] on resource scheduling in HPC. However, these works cannot meet the stream processing requirements. The workload of streaming applications varies dynamically over multiple time scales. It requires the use of sophisticated scheduling and scaling mechanisms.

In order to process streaming data in a timely way, data science community have developed many queries-oriented systems, e.g., TelegraphCQ, STREAM, and Aurora, based on the queries model [14]. However, many of them do not apply to cloud based setting, in which computing is executed by mature parallelism.

To process data streams in cloud and meet the real-time constraints, researchers focus on operator-based data stream processing systems, e.g., Apache Storm, Heron, and IBM Infosphere. But these operator-based systems often lead to overprovision resources to operators. Therefore, [3] proposed a cloud based operator data stream processing system, solving this problem in single streaming application scenario. It lacks built-in support and optimization when the input data streams are shared among multiple parallel applications. We focus on scenarios that are more general. In the context of cloud computing, resource management is the process of allocating computing resources to multiple applications [15], in a manner that seeks to jointly meet the performance objectives of a set of applications. Besides, numerous streaming applications also bring Big Data challenges.

Thus, we propose HPC2-ARS to overcome these challenges. It equips a self-configuring and self-optimizing resource-scheduling module, enabling elastic scheduling services for multiple parallel streaming applications.

III. Overview of Approach

Given the operator topologies (computing logic) of the streaming applications in a system, the current resource allocation, and characteristics of the data streams. Our approach aims to: (a) estimate and minimize resource consumption in real-time stream analytic; (b) estimate and minimize the *average sojourn time*, i.e., the duration from an input tuple first arrives at the system until it is fully processed.

A. Performance Model

In this section, we present a performance model for HPC2-ARS, which estimates query response time given a resource allocation scheme. To model the relationship between the system's performance and the resource consumption of steaming applications, the crucial parameters during the process of input tuples are investigated: the input tuples arrival rate and the processing ability of streaming applications. This ability is decided by the amount of processors and the operator topology. In the topology, we presume that the operators have achieved *load balancing*, which means the processors assigned to the same operator implement processing work equally.

After reviewing various literatures and testing them through empirical studies, we build the performance model based on the Jackson network (*M/M/e* Open Queueing Network). The performance of the system is indicated by the *average sojourn time* of one input tuple in the system. To simplify the presentation, the *average sojourn time* is represented by symbol *T*, the objective is to estimate *average sojourn time E[T]*. The sojourn time of one input tuple of an streaming application includes the processing time of the tuple and its intermediate results, super add queueing delays in the topology.

Presume that there are Q (integer) streaming applications in the system, and each application topology includes certain amount of operators. The number of operators in each application is modeled by a vector $\boldsymbol{c} = (c_1, c_2, ..., c_Q)$, where $\boldsymbol{c}_i$ $(1 \leqslant i \leqslant Q)$ corresponds to the number of operators designed for the i-th application in the system. And the resource allocation in the i-th application is modeled by vector $\boldsymbol{e}_i = (e_{i1}, e_{i2}, ..., e_{ici})$, where e_{ij} $(1 \leqslant i \leqslant Q;\ 1 \leqslant j \leqslant c_i)$ corresponds to the amount of processors assigned to the j-th operator of the i-th application in the system. Therefore, the system resource configuration could be represented by a matrix $\boldsymbol{rc} = (\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_Q)$. Inside the *i-th* application, the sojourn time of one tuple at the *j-th* operator is represented by sysbom T_{ij}. In addition, the operators can be modeled as an $M/M/e_{ij}$ system, where e_{ij} is the amount of processors allocated to the *j-th* operator of the *i-th* application in the parallel system. The sojourn time estimation of an input tuple at an operator is calculated by (1):

$$E[T_{ij}](e_{ij}) = \begin{cases} \dfrac{(\frac{\lambda_{ij}}{\mu_{ij}})^{e_{ij}} \left[\sum_{l=0}^{e_{ij}-1} \frac{(\frac{\lambda_{ij}}{\mu_{ij}})^{l}}{l!} + \frac{(\frac{\lambda_{ij}}{\mu_{ij}})^{e_{ij}}}{e_{ij}!\left(1-\frac{\lambda_{ij}}{\mu_{ij}e_{ij}}\right)} \right]^{-1}}{e_{ij}(1-\frac{\lambda_{ij}}{\mu_{ij}e_{ij}})^2 \mu_{ij} e_{ij}} + \dfrac{1}{\mu_{ij}}, for\ \ e_{ij} > \dfrac{\lambda_{ij}}{\mu_{ij}} \\ +\infty \qquad\qquad , for\ \ e_{ij} \leq \dfrac{\lambda_{ij}}{\mu_{ij}} \end{cases} \tag{1}$$

In (1), $E[T_{ij}](e_{ij})$ is a function of e_{ij} (the amount of processors in one operator), λ_{ij} represents the mean arrival rate of input tuples at the *j-th* operator of *i-th* application in the system, μ_{ij} represents the mean processing speed of the processors. As we can see for (1), the sojourn time $E[T_{ij}]$ of an operator includes two components: the queuing delay and the processing time $1/\mu_{ij}$. To keep the operator running, it need to be assigned enough processors to handle the fast

input tuples rate λ_{ij}.

To estimate, $E[T_i](\boldsymbol{e_i})$, the sojourn time of an input tuple of the *i-th* application in the system, summing up all $E[T_{ij}](e_{ij})$. Specifically, summing up the weighted average of $E[T_{ij}](e_{ij})$.

$$E[T_i](\boldsymbol{e_i}) = E[T_{ij}](e_{i1}, e_{i2}, \dots, e_{ic_i}) = \frac{1}{\lambda_{i0}} \sum_{j=1}^{c_i} \lambda_{ij} E[T_{ij}](e_{ij}) \quad (2)$$

In (2), λ_{i0} stands for the external arrival rate of input tuples flowing into the topology of the *i-th* application in the system. To achieve real-time processing, the system need to assign at least $\gamma_{\min}$ processors to the streaming application. $\gamma_{\min}$ can be calculated by summing up all e_{ij} of the *i-th* application in the system.

Finally, as we are scheduling resources for numerous parallel streaming applications, the weighted average of $E[T_i](\boldsymbol{e_i})$ is used to indicate real-time processing level of the system as follow:

$$E[T](\boldsymbol{c}) = \frac{1}{Q\lambda_s} \sum_{i=1}^{Q} \lambda_{i0} E[T_i](\boldsymbol{e_i}) \quad (3)$$

Equation (3) accomplished the HPC2-ARS performance model, λ_s is the mean arrival rate of the Big Data Streams to the system, Q is the amount of streaming applications.

B. Scheduling Strategy

We designed a strategy for autonomous resource scheduling, inspired by Olympic spirit (faster). The scheduling algorithm based on this strategy makes resource allocation decisions to assign processors to different applications (operators).

This strategy dedicated to achieve the fastest processing speed. It makes HPC2-ARS capable of improving the real-time processing level given adequate resources, or derive a sub-optimal solution if the amount of resource R_{max} is inadequate to achieve real-time processing. The motivation is to try the system's best to achieve the fastest processing speed on Big Data Streams analytic. It could be modeled by the following optimization problem (4).

$$\min_{e_{ij}} E[T](\boldsymbol{c}), \quad \text{s.t. } \sum_{i=1}^{Q} \gamma_{min}(i) \leq R_{max} \quad (4)$$

The algorithm starts from the smallest possible value of each e_{ij} (lines 1-5). Then it checks the status of resource usage (lines 6-8). It iteratively adds one processor to the operator which leads to the largest decrease in *average sojourn time* $E[T](\boldsymbol{c})$, until out of resources (lines 9-19). At last, if $E[T](\boldsymbol{c}) <= T_{max}$, HPC2-ARS outputs an optimal resource configuration that improves the real-time performance level, else it outputs an optional sub-optimal solution (lines 20-24).

Input: R_{max}, λ_s, $\boldsymbol{c} = (c_1, c_2, \dots, c_Q)$, $\{\lambda_{i0}, i = 1,\dots,Q\}$, $\{\lambda_{ij}, i = 1,\dots,Q, j = 1,\dots, c_i\}$, $\{\mu_{ij}, i = 1,\dots,Q, j = 1,\dots, c_i\}$

Output: $rc = (e_1, e_2, \dots, e_Q) = (e_{11}, e_{12}, \dots, e_{1c_1}; e_{21}, e_{22}, \dots, e_{2c_2}; e_{Q1}, e_{Q2}, \dots, e_{Qc_Q})$, $E[T](\boldsymbol{c})$

1: **for all** $i \leftarrow 1, \dots, Q$ **do**
2: **for all** $j \leftarrow 1, \dots, c_i$ **do**
3: $e_{ij} \leftarrow \left\lfloor \frac{\lambda_{ij}}{\mu_{ij}} \right\rfloor + 1$ /* initialize each e_{ij} */
4: **end for**
5: **end for**
6: **if** $\sum_{i=1}^{Q} \sum_{j=1}^{c_i} (e_{ij}) > R_{max}$ **do**
7: Throw an exception that the resource request exceeds the system's limitation.
8: **end if**
9: **while** $\sum_{i=1}^{Q} \sum_{j=1}^{c_i} (e_{ij}) \leq R_{max}$ **do**
10: **for all** $i \leftarrow 1, \dots, Q$ **do**
11: **for all** $j \leftarrow 1, \dots, c_i$ **do**
12: $\omega_{ij} \leftarrow E[T](e_{ij}) - E[T](e_{ij} + 1)$
13: **end for**
14: **end for**
15: /* find the topology position which leads to the largest decrease in sojourn time $E[T](\boldsymbol{c})$*/
16: $i' \leftarrow \arg max_{ij}\, \omega_{ij}$
17: $j' \leftarrow \arg max_{ij}\, \omega_{ij}$
18: $e_{i'j'} \leftarrow e_{i'j'} + 1$
19: **end while**
20: **if** $E[T](\boldsymbol{c}) \leq T_{max}$ **do**
21: **return** rc, $E[T](\boldsymbol{c})$
22: **else do**
23: Throw an exception that T_{max} is not achievable and a sub-optimal solution is optional.
24: **end if**

IV. System Architecture

HPC2-ARS leverage Tianhe-2 supercomputer, Kylin Cloud and stream processing frameworks (e.g., Storm and Heron), providing high performance streaming services. It is a distributed fault-tolerant real-time computation architecture. It answers for performance measurement, resource scaling and allocation. Its implementation is challenging, we need to bridge the gaps among supercomputer infrastructure, cloud computing, Apache stream processing systems, and the abstract optimization approach.

A. Tianhe-2 supercomputer

Tianhe-2 provides high performance computing infrastructure. Multi-core platforms, like supercomputer, have proven themselves capable to accelerate numerous data-intensive applications [17]. We carry out the Big Data Streams analytic on Tianhe-2, the second generation of massively parallel supercomputers in the TH series. It is capable of a peak performance of 54.9 Pflops and achieves a sustained performance of 33.9 Pflops. Tianhe-2 employs accelerator-based architectures. Each compute node is equipped with 2-way 12-core Intel Xeon E5-2692 v2, three Intel Xeon Phi accelerators based on the many-integrated-core (MIC) architecture and 128G memory, delivering a peak performance of 3.432 teraflops. All computing nodes are connected with a customized network TH-Express 2. The bi-directional bandwidth of TH-Express 2 can achieve 20GB/s. The system is packaged in a compact structure. Each compute rack has four compute frames, and each frame contains one switch board, one monitor board, and 32 compute nodes packaged in 16 compute broads. The system can be maximally configured with 144 racks, up to 18304 compute nodes.

B. Kylin Cloud

Kylin Cloud provides cloud-based resource pool service. Ultimately, Big Data Streams analytic can profit from supercomputer developments only if software can be made to fit. Tianhe-2 has a software stack designed to accelerate data-intensive applications. One very important component

of the software stack is Kylin Cloud that is customized from OpenStack based on Tianhe-2 infrastructure. Kylin Cloud platform has been deployed on about 6400 nodes of Tianhe-2, with 256 nodes as the control nodes of the cloud platform and the remaining nodes as the compute nodes running virtual machines. It mainly provide IaaS through virtualization technology. All kinds of physical resources are used to build a virtual resource pool. It provides Apache stream processing systems with a certain amount of resources in the form of virtual machines that serve as cloud servers with continuous availability and scalability. It hides complex computing infrastructures and provides flexible specification of Tianhe-2 with details. So that to efficiently leverage the Tianhe-2 hardware platform, scaling to accommodate Big Data Streams on Apache stream processing systems, bridging the increasing gap between the growth of streaming data and computing power.

C. Autonomous Resource Scheduling Modules

It is of great important to assemble the precise measurement from the HPC2 and aggregate the statistics. There are configuration reader module and measurer module to generate the statistics, which is needed by the optimizer, based on the data flow and control flow from HPC2. The configuration reader module is drafted to be an interface for controlling a data structure that contains the configuration parameters provided by either the streaming applications or HPC2. The measurer module is responsible for the measurement on HPC2 and to preprocess the measurement before messaging them to the optimizer. Utilizing measurer module, we are able to get essential metrics, e.g., various inputs arrival rates, and the processing rates, etc.

HPC2-ARS also need to make online decisions (how many processors should be assigned to the extraordinary applications/operators) and control the HPC2 resource allocation. To transform the decisions into executable commands for HPC2, there are resource negotiator and scheduler module concluding an effective mapping between the available processors and the applications (operators). And when the applications submit new configurations to request more processors, the resource negotiator module will interact with the lower layer resource managers of HPC2.

V. CONCLUSION

In summary, to deal with Big Data Streams in real-time, we devised HPC2-ARS. It promises increased elasticity and efficiency for streaming services, provisioning computational resources autonomously on demand, and satisfying the requirements of real-time and stream processing of Big Data.

ACKNOWLEDGMENT

This research was supported in part by NSFC-Guangdong Joint Found (U1501254), Natural Science Foundation of China (61472089), NSFG (2014A030306004, 2014A030308008), Science and Technology Planning Project of Guangdong (2015B010108006, 2015B010131015), Guangdong High-level Personnel of Special Support Program (2015TQ01X140), Pearl River S&T Nova Program of Guangzhou (201610010101), and STPPG (201604016075).

REFERENCES

[1] Padgavankar, M. H., & Gupta, S. R. (2014). Big Data storage and challenges. International Journal of Computer Science & Information Technolog.

[2] Chen, C. L. P., & Zhang, C. Y. (2014). Data-intensive applications, challenges, techniques and technologies: a survey on Big Data. Information Sciences, 275(11), 314-347.

[3] Fu, T. Z. J., Ding, J., Ma, R. T. B., Winslett, M., Yang, Y., & Zhang, Z. (2015). DRS: Dynamic Resource Scheduling for Real-time Analytic over Fast Streams. IEEE, International Conference on Distributed Computing Systems (Vol.690, pp.411-420). IEEE.

[4] Kambatla, K., Kollias, G., Kumar, V., & Grama, A. (2014). Trends in Big Data analytic. Journal of Parallel & Distributed Computing, 74(7), 2561-2573.

[5] Khan, M., Li, M., Ashton, P., Taylor, G., & Liu, J. (2014). Big Data analytic on PMU measurements. International Conference on Fuzzy Systems and Knowledge Discovery. IEEE. (IEEE Transactions)

[6] Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M., & Herrera, F. (2017). A survey on data preprocessing for data stream mining: current status and future directions. Neurocomputing, 239, 39-57.

[7] Liao, X., Xiao, L., Yang, C., & Lu, Y. (2014). Milkyway-2 supercomputer: system and application. Frontiers of Computer Science, 8(3), 345-356.

[8] Assunção, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A., & Buyya, R. (2015). Big Data computing and clouds: Trends and future directions. Journal of Parallel and Distributed Computing, 79, 3-15.

[9] Rehr, J. J., Vila, F. D., Gardner, J. P., Svec, L., & Prange, M. (2010). Scientific computing in the cloud. Computing in Science & Engineering,12(3), 34-43.

[10] Q. Zhang, L. Cheng, & R. Boutaba. (2010). Cloud computing: state-of-the-art and research challenge. Journal of Internet Services and Applications. 1(1), 7-18. (Conference proceedings)

[11] Idris, M., Hussain, S., Ali, M., Abdulali, A., Siddiqi, M. H., & Kang, B. H., et al. (2015). Context‐aware scheduling in mapreduce: a compact review. Concurrency & Computation Practice & Experience, 27(17), 5332-5349.

[12] Jiang, H. J., Huang, K. C., Chang, H. Y., Gu, D. S., & Shih, P. J. (2011). Scheduling Concurrent Workflows in HPC Cloud through Exploiting Schedule Gaps. Algorithms and Architectures for Parallel Processing, International Conference, Ica3pp, Melbourne, Australia, October 24-26, 2011, Proceedings (Vol.7016, pp.282-293). DBLP.

[13] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., & Evans, R., et al. (2013). Apache Hadoop YARN:yet another resource negotiator. Symposium on Cloud Computing (pp.1-16). ACM.

[14] Panigati E., Schreiber F.A., Zaniolo C. (2015) Data Streams and Data Stream Management Systems and Languages. In: Colace F., De Santo M., Moscato V., Picariello A., Schreiber F., Tanca L. (eds) Data Management in Pervasive Systems. Data-Centric Systems and Applications. Springer, Cham

[15] Jennings, B., & Stadler, R. (2015). Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management, 23(3), 567-619.

[16] Harel, A., Francis, J. C., & Harpaz, G. (2018). Alternative utility functions: review, analytic and comparison. Review of Quantitative Finance and Accounting, 1-27.

[17] Amesfoort, A. S. V., Varbanescu, A. L., Sips, H. J., & Nieuwpoort, R. V. V. (2009). Evaluating multi-core platforms for HPC data-intensive kernels. ACM Conference on Computing Frontiers (pp.207-216). ACM.