Hardware-Aware In-Memory Data Processing Systems

Chen Luo

1 Introduction

Large-scale Big Data analytics is becoming more and more critical for today's business, and lots of efforts have been made to improve the performance of data processing systems. On one hand, larger volume of data are being produced in a even faster rate. On the other hand, the advancement of modern hardware provides further opportunities for performance improvement. For example, in-memory data processing can significantly improve response time and throughput [1]. From the user's prospective, it would be ideal to simply upgrade the hardware configurations to speedup data processing systems. However, this is often challenging in practice. As the recent work [2] shows, naively using advanced hardware would not necessarily bring significant performance gains for today's data management and processing systems. Thus, this motivates the *hardware-aware in-memory data processing system* to fully unleash the power of modern hardware capabilities.

Developing a general purpose hardware-aware data processing system is challenging. Besides the requirements for traditional data processing systems, e.g., scalability and fault-tolerance, the proposed system should also address the following additional requirements/challenges:

1. Hardware-awareness: Modern hardware architectures, like graphics processing units (GPU), non-uniform memory access (NUMA) architecture, non-volatile memories (NVM) and remote direct memory access (RDMA) network, could provide significant performance improvement, but only when operated properly. For example, NUMA architecture has non-uniform memory access rate (local memory vs. remote memory), which requires proper data placement/access strategies to achieve better performance. Thus, as a basic requirement, the system must be aware of the characteristics of modern hardware architectures to make full use of their power.
2. One-size-fits-many: While a specific algorithm can be bound to certain hardware architecture, the system must not. To be a general purpose data processing system, the proposed system should be able to work with different hardware configurations, i.e., one size fits many. More specifically, the proposed system should be capable of identifying the hardware configurations of the underling cluster, and automatically choose the optimal (or near-optimal) plan to process the query. One should also keep in mind that the cluster is usually heterogeneous in practice, which poses further challenge for the system.
3. Extensibility: Due to the technical advancements of hardware industries, hardware capabilities may change quickly, e.g., lower cost per bit for NVM, more CPU cores and even new architectures. The handle such changes, the system requires novel architectural design to be extensible so that the newly emerged hardware can be easily integrated/plugged into the system.

The remaining of the proposal is organized as follows. Section. 2 briefly reviews the related works. Section. 3 proposes the preliminary design of the system and discuss how the above requirements are addressed. Finally, Section. 4 concludes the proposal.

2 Related Works

In this section, we briefly review the state-of-the-art works on hardware-aware data processing.

To embrace modern hardware architectures, many hardware-aware query processing algorithms have been proposed [3,4,5]. The work [3] considers concurrent scan on columnar-stores on NUMA architecture. Claude et at. [5] studies the problem of in-memory joins using RDMA network. Join et al. [4] proposes a query processing technique on CPU-GPU architectures. All these works confirm that proper use of modern hardware architecture could lead to significant performance improvement. However, these works only satisfy the requirement 1, i.e., hardware-awareness, but fail to achieve the requirements 2 and 3 due to the fact that they are bound to certain hardware architecture.

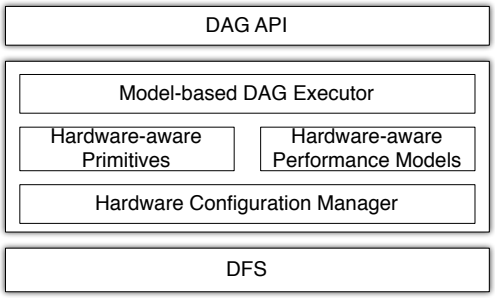

Fig. 1: Architectural Design of the Proposed System

Besides the hardware-aware data processing algorithms, a lot of efforts have been made to develop efficient in-memory data processing systems [6,7,8] M3R [6] is an in-memory extension of MapReduce [9], but mainly targets at short-running jobs on high mean-time-to-failure clusters. Spark [7] is a general-purpose in-memory data processing system, and provides a data abstraction called Resilient Data Set (RDD). The work [8] extends MapReduce to support in-memory efficient stream data processing. These systems only partially address the requirements 1 and 2, mainly for their in-memory and concurrent processing. However, many advanced hardware architectures, like NUMA and RDMA, are still unexploited yet, and it is unclear how to plug new hardware architectures effectively into these systems (requirement 3).

3 Research Design

As discussed in the previous section, to the best of our knowledge, none of the existing works addresses the three requirements simultaneously. In this section, we present the architectural design of the proposed hardware-aware in-memory data processing system, and briefly discuss the requirements are addressed.

3.1 Architectural Design

The architectural design is shown in Fig. 1. Due to space limitation, we mainly discuss the components pertaining to the proposed system, while leave out the common components, e.g., master/ worker architecture, distribute file system and the DAG computation model.

Hardware Configuration Manager. In order for the system to percept the underling cluster, the hardware configuration manager collects and manages the hardware information of all workers. These information include but are limited to the memory sizes, number of CPU cores, memory/network read/write speed, the available hardware architectures (e.g., NUMA, RDMA, GPU, NVM) for each work.

Hardware-aware Primitives. Each primitive can be viewed as a hardware-specific data processing operator, and the primitives together constitute the building blocks of the proposed system. For example, there could be a hash join operator for normal network setting, and a join operator specially designed for RDMA network. These primitives include but are not limited to hardware-aware indexing, scan, join, group, aggregation etc.

Hardware-aware Performance Models. For each hardware-aware primitive, there should be a corresponding performance model, which is used for the DAG executor to select the optimal execution plan. These models could be parameterized with certain parameters, e.g., memory sizes, memory/network read/write speed and certain statistics of the dataset. Note that these parameters may vary for different hardware architectures.

Model-based DAG executor. With the performance models and the hardware configuration information of the clusters, the model-based DAG executor generates an optimal execution plan for the (logical) DAG task. More specially, the executor formulates an optimization problem with the available hardware information, performance models and statistics of the dataset, and tries to find an optimal execution plan by solving the optimization problem. The execution plan assigns certain workers to process certain input data with certain primitives, and shuffle the data when necessary

with the chosen shuffling algorithm. Moreover, the executor could also change the execution plan dynamically, and use approximate optimal plan to ensure timely response.

3.2 Addressing the Requirements

We then briefly discuss how the proposed system addresses the three requirements.

The requirement 1 of hardware-awareness is naturally addressed by the hardware-aware primitives since each primitive corresponds to a specific hardware architecture to fully unleash its power. For the requirement 2 of one-size-fits-many, the proposed system automatically collects the hardware information of the underling cluster, and choose the optimal execution plan based on the hardware configurations. In other words, although the primitives are bound to certain hardware architecture, the system would try to find suitable primitives for the cluster so that the system is not bound to any specific hardware.

Finally, for the requirement 3 of extensibility, the proposed system could be easily adapted for the hardware changes. Note that hardware could change in two ways, namely capacity changes or new hardware architectures. For the first case, the changed capacity would be (ideally) automatically captured by the hardware configuration manager and the parameterized performance model. While for new hardware architecture, the specific primitives should be developed and the corresponding performance models should be provided. Moreover, the information of the new architecture should also be registered to the hardware configuration manager. In practice, the maintenance would not cause too much effort, since the second case usually happens less frequently.

4 Conclusion

In this proposal, we discuss the requirements of hardware-aware data processing systems and review the current research status on this area. We further propose an architectural design to address these requirements. To make the system possible, a lot of efforts need to be made, including novel hardware-aware data processing primitives, performance models, and model-based DAG executor. Moreover, extensive experiments should be performed to compare the proposed system with the existing systems, to study the maintenance effort of plugging new hardware and study the performance of the system in different and heterogeneous clusters.

References

1. Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang. In-memory big data management and processing: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 27(7):1920–1948, July 2015.
2. Kian-Lee Tan, Qingchao Cai, Beng Chin Ooi, Weng-Fai Wong, Chang Yao, and Hao Zhang. In-memory databases: Challenges and opportunities from software and hardware perspectives. *SIGMOD Rec.*, 44(2):35–40, August 2015.
3. Iraklis Psaroudakis, Tobias Scheuer, Norman May, Abdelkader Sellami, and Anastasia Ailamaki. Scaling up concurrent main-memory column-store scans: Towards adaptive numa-aware data and task placement. *Proc. VLDB Endow.*, 8(12):1442–1453, August 2015.
4. Jiong He, Shuhao Zhang, and Bingsheng He. In-cache query co-processing on coupled cpu-gpu architectures. *Proc. VLDB Endow.*, 8(4):329–340, December 2014.
5. Claude Barthels, Simon Loesing, Gustavo Alonso, and Donald Kossmann. Rack-scale in-memory join processing using rdma. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1463–1475, New York, NY, USA, 2015. ACM.
6. Avraham Shinnar, David Cunningham, Vijay Saraswat, and Benjamin Herta. M3r: increased performance for in-memory hadoop jobs. *Proceedings of the VLDB Endowment*, 5(12):1736–1747, 2012.
7. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. 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*, pages 2–2. USENIX Association, 2012.
8. Boduo Li, Edward Mazur, Yanlei Diao, Andrew McGregor, and Prashant Shenoy. A platform for scalable one-pass analytics using mapreduce. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 985–996. ACM, 2011.
9. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.