The Intersection of Big Data and HPC

Using Asynchronous Many Task Runtime Systems for HPC and Big Data

Joshua Suetterlein
Pacific Northwest National Laboratory
Richland, Washington
joshua.suetterlein@pnnl.gov

Joshua Landwehr
Trovares, inc
Seattle, Washington
snaphat@gmail.com

Andres Marquez
Pacific Northwest National Laboratory
Richland, Washington
andres.marquez@pnnl.gov

Joseph Manzano
Pacific Northwest National Laboratory
Richland, Washington
joseph.manzano@pnnl.gov

Kevin J Barker
Pacific Northwest National Laboratory
Richland, Washington
kevin.barker@pnnl.gov

Guang R Gao
University of Delaware
Newark, Delaware
ggao.capsl@gmail.com

ACM Reference format:
Joshua Suetterlein, Joshua Landwehr, Andres Marquez, Joseph Manzano, Kevin J Barker, and Guang R Gao. 2017. The Intersection of Big Data and HPC. In *Proceedings of SC 17, Denver, Colorado, USA, November 2017 (SC17),* 4 pages.
https://doi.org/

1 INTRODUCTION

As recently established, Big Data has positioned itself as the fourth paradigm of science. Big data uses large clusters to turn enormous volume of data into insight. Meanwhile, High Performance Computing (HPC) uses large clusters to run large scientific simulation of natural or human-made phenomena. Although their primary objectives seem different, HPC suffers from a growing size of its workloads and its associated challenges. This creates interesting research challenges for both fields on how to effectively use big data techniques on HPC workloads and how to map big data frameworks and algorithms onto emerging HPC platforms. This poster presents a case study which uses Asynchronous Many Task Runtimes (AMTs) as exploratory vehicles to highlight possible solutions to the above questions. AMTs presents the unique opportunity of using smaller units of work (better load balancing and more adaptable), reconfigurable schedulers and data layouts that can be connected to introspection frameworks, and an ability to produce and exploit a massive amount of concurrency. We use the Performance Open Community Runtime (P-OCR) as a vehicle to port MapReduce operators to the HPC realm. We conduct experiments for both strong and weak scaling experimental format using WordCount and TeraSort as our kernels. The experimental results showcase promising results for both weak and strong scaling with the strong scaling ones showcasing hyper-linear speedups and the weak scaling experiments exhibiting a maximum degradation of 50% in the worst case. We continue this paper by explaining the exemplar runtime used in this effort: P-OCR.

SC17, November 2017, Denver, Colorado, USA

ACM ISBN . . . $15.00
https://doi.org/

2 PERFORMANCE OPEN COMMUNITY RUNTIME

The Open Community Runtime is a standard for an asynchronous, event-driven, fine-grain execution model. The Performance Open Community Runtime (P-OCR) is an implementation of the OCR standard version 0.99a. The P-OCR's API supports the creation of Event Driven Tasks (EDTs), events, and Data Blocks (DBs) and synchronization via a signaling API. In addition, P-OCR's API includes support Global Unique Identifiers for all of its primitives, synchronized parallel start, direct EDT signaling, an introspective and adaptive framework and an extended memory model. API calls are transparently resolved into local or remote operations depending on the operation and the locality of the operands. Local operations include the creation of runtime objects (i.e. EDTs, events, and DBs) and the signaling of local EDTs and events. Remote operations include the signaling of EDTs and events not resident on the signaling node and the acquisition of remote DBs. Determining the locality of a runtime object is achieved using a GUID and a distributed routing table along with a coherency protocol.

To communicate across nodes, P-OCR is supported by several backends include TCP, MPI, and Rsockets [1]. P-OCR employs a packed structure packet protocol which assumes a homogeneous architecture, and can copy a structure without marshalling and demarshalling overhead. This can be easily extended to support heterogeneous system with a packing/serialization protocol similar to [2].

3 BIG DATA OPERATORS

To support Big Data in P-OCR, we implement four types of streaming MapReduce-like operators including mappers, shuffles, reducers, and local reducers. These operators are joined together to form a chain. An operator in a chain will evaluate some number of Key / Value (K/V) pairs contained in an iterable DB[1]. Both the mappers and local reducers use thresholds to adjust the number of K/V pairs evaluated at a time (i.e. size of the stream). Upon completion, the operator generates a new iterable DB of K/V pairs to send to the next operator. The execution of a chain continues until each

[1] special Big Data Datablock in P-OCR

operator has processed all of the K/V pairs, and a completion signal is propagated. In addition, the K/V pairs are programmatically partitioned, and the affinity of partition is configurable.

A common chain is typically comprised of a mapper, local reducer, shuffle, and reducer. The streaming mapper is used to parse, transform, filter, or ingest data from the file system. Local reducers are then used to aggregate and compress K/V pairs belonging to the same partition prior to a network shuffle. An explicit shuffle operator is used to aggregate K/V pairs on a single node (unlike HadoopâĂŹs implicit shuffle at the end of each mapper). Finally, reducers are used to distill the results by combining the K/V pairs from a single partition on a single node.

P-OCR employs two forms of parallelism, chain and inter-chain parallelism. Chain parallelism is similar to the partition parallelism found in popular Big Data solutions. In P-OCR, an instance of a chain is executing on each node to evaluate distinct partitions of K/V pairs. Inter-chain parallelism is similar the parallelism exposed via work stealing frameworks such as Cilk. The streaming operators in P-OCR are evaluated depth-first, and the granularity of each operator execution is dependent on its threshold or input iterable DB. An operator will suspend its execution upon reaching a threshold, and begins evaluating the next operator in the chain. The suspended operatorâĂŹs execution can then be stolen and executed concurrently facilitating load balancing across a single node. Furthermore, P-OCR explores the trade-off of parallelism versus task granularity by adjusting operatorsâĂŹ thresholds leveraging introspection and adaption.

4 EXPERIMENTAL TESTBED

For our exploratory study, we used the Pacific Northwest National Laboratory PAL cluster which is comprised of 128 nodes connected by an Infiniband FDR network [3]. Each node contains two AMD Opterons 6272 each with 16 cores running at 1.4 GHz. The node is equipped with 2 MB of cache and 64 GB of DRAM. In the presented experiments all threads are pinned to different cores. Both P-OCR and the benchmarks were compiled using GCC 4.8. The TeraSort and WordCount were chosen as representative kernels based on their importance to the Big Data Community (as building blocks) and its relative ease to port to new systems. We run the benchmarks in both strong and weak scaling modes to showcase the performance (expressed as relative speedup) with data sizes up to 2 Terabytes for Word Count and up to half a terabyte in the case of TeraSort. We also ran two sets of experiments for the largest weak scaling mode in which we compare the runtime behavior of both adaptive and non-adaptive modes of the P-OCR infrastructure.

5 IN-DEPTH CASE STUDIES

Table 1a presents the relative speedup for both weak and strong scale studies for Word Count. We use the best configuration empirically found (128 K/V pairs) for both the mapper and network threshold. The super-linear speedup comes from the ability to ingest files faster using the lustre file system. We see similar behavior while weak scaling and it can be seen when looking at the heatmaps presented in the poster. In the weak scaling experiments, we process process 32 GB per node, totaling 2 TB for the 64 node case.

Table 1b presents the relative speedup for both weak and strong scaling for the TeraSort. The large speedup is a result of the red black tree implementation. As the number of nodes increases, so does the number of partitions. Since there are less entries per partition, the insertion time decreases (the insertion complexity of red black tree is O(log(n))). Assuming the data is roughly distributed evenly among partitions, we will see a super linear speedup event without considering any caching effects. The remaining speedup is attributed to caching effects and increased file reading time from lustre. The reason for not using a fixed large number of partitions for all of the node configurations is the additional memory overhead can be large and limit the problem size.

Nodes	Strong	Weak
1	1	1
2	2.29	1.15
4	4.76	1.19
8	9.64	1.19
16	19.36	1.19
32	36.86	1.15
64	66.74	1.05

(a) Word Count.

Nodes	Strong	Weak
1	1	1
2	1.46	1.23
4	28.39	1.24
8	55.42	1.52
16	107.74	1.18
32	201.36	1.09
64	310.75	0.88

(b) TeraSort.

Table 1: Relative Speed up for Strong and Weak Scaling Studies

For the weak scaling experiments, at 64 nodes, the benchmark shows a 12% decrease in performance. The TeraSort benchmark places an immense pressure on the network since none of the data is ever reduced. Instead the data is shuffled as a whole around the entire system. At 64 nodes, we see the network pressure dominating performance (as showcased in the heatmaps in the poster).

5.1 Adaptive versus Best Empirical Based Runs

The WordCount heatmaps presented in the "Word Count Case Study" section of the poster shows a comparison between adaptation and best empirical found size for the weak scaling experiment running on 64 nodes processing 2 TB across all nodes. For the best empirical based configuration, we notice a similar pattern as before (i.e. the execution is once again dominated by the network). Using the adaption capabilities of P-OCR, the pressure shifts from the network to file ingestion. In this case however, we see the rate of ingestion is not constant across all nodes due to contention on the lustre file system. This behavior is exacerbated by increasing the number of nodes and keeping the lustre file stride constant at 8. Due to this behavior, the adaptive framework experiment exhibits a 6% degradation in comparison to the best empirical run.

The heat maps for TeraSort on 64 nodes processing 465 GB across are presented in the poster under the TeraSort Case Study section (cf Poster). For the execution without adaption, the file ingestion is sluggish and more infrequent. The network is under utilized until the ingestion is almost completed which creates cascading events of inefficiency across the application. Conversely, the execution with adaption speeds up the file ingestion, but creates more contention on the network that leads to a slower execution.

6 CONCLUSIONS

From our case studies we see the appropriate granularity is not one-size-fits-all. Rather it can change for the node configuration, size of input data, and the dataset itself making effective adaptive frameworks are necessity for these fields.

While AMTs can effectively support Big Data, these applications push AMT's to their limits exposing contention (in the file system and network) to be the limiting factors in scalable performance.

REFERENCES

[1] Sean Hefty. 2013. RSOCKETS. http://downloads.openfabrics.org/Media/Monterey_2012/2012_Workshop_Mon_Rsockets.pdf. (2013).
[2] Christopher Lauderdale, Robert Springer, and Rishi Khan. [n. d.]. An execution model for adaptive load-balancing on multicore and multi-GPU systems. ([n. d.]).
[3] Qian Liu and Robert D. Russell. 2014. A Performance Study of InfiniBand Fourteen Data Rate (FDR). In *Proceedings of the High Performance Computing Symposium (HPC '14)*. Society for Computer Simulation International, San Diego, CA, USA, Article 16, 10 pages. http://dl.acm.org/citation.cfm?id=2663510.2663526