September 2019

Characterization of Power Usage and Performance in Data-Intensive Applications using MapReduce over MPI

Joshua DAVIS [a], Tao GAO [a], Sunita CHANDRASEKARAN [a], Heike JAGODE [b], Anthony DANALIS [b], Jack DONGARRA [b], Pavan BALAJI [c], and MichelaTAUFER [b,1]

[a] *University of Delaware*
[b] *University of Tenneesee Knoxville*
[c] *Argonne National Laboratory*

Abstract. This paper presents a quantitative evaluation of the power usage over time in data-intensive applications that use MapReduce over MPI. We leverage the PAPI powercap tool to identify ideal conditions for execution of our mini-applications in terms of (1) dataset characteristics (e.g., unique words in datasets); (2) system characteristics (e.g., KNL and KNM); and (3) implementation of the MapReduce programming model (e.g., impact of various optimizations). Results illustrate the high power utilization and runtime costs of data management on HPC architectures.

Keywords. Data management, KNL, KNM, PAPI, Combiner optimizations

1. Introduction

The contributions of this paper are in the growing high performance computing (HPC) field of data analytics and are at the cross-section of empirical collection of performance results and the rigorous, reproducible methodology for their collection. Our work expands traditional metrics such as execution times to include metrics such as power usage and energy usage associated with data analytics and data management. We move away from the traditional compute-intensive workflows towards data-intensive workloads with a focus on MapReduce programming models as they gain momentum in the HPC community.

Emerging HPC platforms are generally designed with data management and, in particular, the associated power usage in mind. Trends in HP exhibits the widening gap between flops and IO bandwidth peaks. The latter is capped to contain the power usage of the HPC systems. In other words, the need for power capping on these systems has created substantial constraint on the bandwidths with respect to moving data through the memory hierarchy. Overall, it is a common belief that data management (i.e., process-

[1] Corresponding Author: Address: Electrical Engineering and Computer Science Dept., The University of Tennessee, Knoxville, TN 37996-2250; E-mail: taufer@utk.edu.

ing data on the core and moving data through the memory hierarchy) is power-intensive. Still, little work is available in providing quantitative evaluations of these costs.

The paper tackles this problem by addressing the need to quantitatively measure the impacts of data management on performance in MapReduce-based applications when executed on HPC systems. Specifically, we present studies on the impact of power capping on performance metrics such as runtime and power usage over time for data-intensive application on top of a MapReduce over MPI framework. The key questions that we look to answer are: Can we determine what data characteristics are the most relevant to the trade-offs between power usage and performance? Can we identify the effects of power capping on the performance of data-intensive applications using the MapReduce programming model on HPC systems? Can we separate the impact of the application itself from the impact of the middleware and the monitoring system on power usage?

2. Testing Environment for Power Measurements

Multilayer Testing Environment: Our testing environment is composed of multiple layers . The middleware layer manages the data analytics. Because we are working on HPC systems, we use Mimir, an open source implementation of MapReduce over MPI, that we enhance with the Performance Application Programming Interface (PAPI) monitoring and capping performance tool. The hardware layer includes two many-core systems that fully support Mimir and PAPI. We target the **MapReduce (MR)** programming model [1] for the data management and analytics because these processes are broadly used and suitable for a wide variety of data applications. Implementations of MapReduce over MPI have gained the most traction in HPC because they provide C/C++ interfaces that are more convenient to integrate with existing scientific applications compared with Java, Scala, or Python interfaces. Moreover, they can use the high-speed interconnection network through MPI. More traditional frameworks such as Hadoop [2, 3] and Spark [4] or tuned versions of these popular MapReduce frameworks on HPC systems [5–8] do not support one or multiple of the following features: they do not provide on-node persistent storage; do not work well (or work at all) on many commodity-network-oriented protocols, such as TCP/IP or RDMA over Ethernet; are not tuned for system software stacks on HPC platforms, including the operating system and computational libraries; or are specialized for a given scientific computing, and hence lack generality. For example, supercomputers such as the IBM Blue Gene/Q [9] use specialized lightweight operating systems that do not provide the same capabilities as those that a traditional operating system such as Linux or Windows might.

We use **Mimir** [10–12] as our MapReduce over MPI framework. Mimir is among the few state-of-the-art, memory-efficient MapReduce over MPI implementations that are open source and support memory efficiency and scalability; reduced synchronizations; reduced data staging and improved memory management; improved load balancing; and capabilities to handle I/O variability. Mimir's in-memory workflow includes a pipeline combiner workflow to compress data before shuffling and reduce operation, using memory more efficiently and balancing memory usage; a dynamic repartition method that mitigates data skew on MapReduce applications without obviously increasing their peak memory usage; and a strategy for splitting single superkeys across processes and further mitigating the impact of data skew, by relaxing the MapReduce model constraints on key partitioning. A MapReduce job traditionally involves three stages: *map*, *shuffle*, and *reduce*. The *map* stage processes the input data using a user-defined map callback function (map operation) and generates intermediate $\langle key, value \rangle$ (KV) pairs. The *shuffle* stage performs an all-to-all communication that distributes the intermediate KV pairs across all processes. In this stage KV pairs with the same key are also merged and stored in $\langle key, \langle value1, value2... \rangle \rangle$ (KMV) lists. The *reduce* stage processes the KMV lists with a user-defined reduce callback function and generates the final output. In Mimir, the shuffling stage consists of two decoupled phases: an aggregate phase that interleaves with the map operations and executes the MPI_AllToAllv communication calls and a convert phase that precedes the reduce operations by dynamically allocating space and assigning KV pairs to list the KVs. The map and reduce operations are implemented by using user callback functions. The aggregate and convert phases are implicit: the user does not explicitly start these phases. This design offers two advantages. First, it breaks global synchronizations between map and aggregate phases and between convert and reduce phases. Mimir determines when the intermediate data should be sent and merged. Mimir also pipelines the four phases to minimize unnecessary memory usage. We still retain the global synchronization between the map + aggregate and convert + reduce phases, which is required by the MapReduce programming model.

We plug **PAPI** [13] into Mimir to set power caps and collect power usage over time, using the powercap interface that is built into the Linux kernel. The purpose of this interface is to expose the Intel RAPL (Running Average Power Limit) [14] settings to userspace, as opposed to directly accessing the RAPL model-specific registers (MSRs), which would require elevated privileges. PAPI provides access to performance counters for monitoring and analysis. We use PAPI to measure the core power/energy consumption, power limit, power of memory accesses, and core frequency. PAPI's powercap component [15] gives access to Intel's RAPL interface, which employs the DVFS technique to apply power limits.

Data-Intensive Miniapplications: We implemented and use three data-intensive miniapplications extracted from the WordCount benchmark: (1) Map+Aggregate, which contains only the map and shuffle phases (without the convert and reduce phases); (2) GroupByKey, which adds the convert/reduce operations for a complete, traditional MapReduce execution; and (3) ReduceByKey, which locally combines KVs with matching keys immediately before shuffling and after shuffling (i.e., combiner optimizations in state-of-the-art MR frameworks).

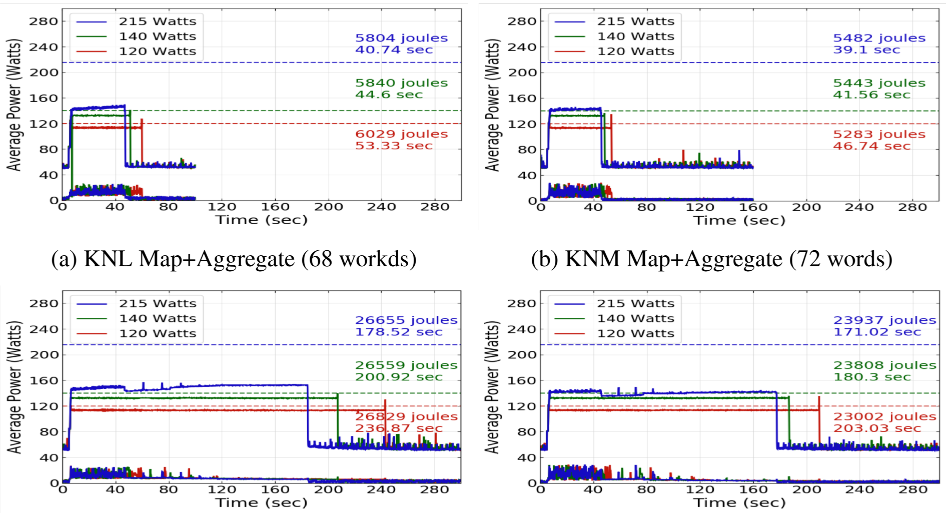
Data Generation: We generate different datasets with a large number of words (4 billion) and a variable number of unique words. In the rest of the paper, each dataset is identified as *XBY*, where *X* is the number of billions of total words and *Y* is the number of unique words repeated in the dataset. Our dataset is larger than available datasets commonly used in benchmarking (e.g., the Wikipedia dataset from the PUMA dataset [16]). The controlled generation of data allows us to create a controlled testing environment that we can use to separate the different factors impacting power consumption in data management. It also enables reproducibility of our results. The Wikipedia dataset with its highly heterogeneous type and length of words would not allow us to handle the same level of controlled testing. The software used for the data generation is open source, as part of the Mimir software [17]. The total number of words defines the overall workload across the processes: the larger the number, the higher the workload per process. The number of unique words determines how the KMV are distributed across processes during shuffle. The number of unique words also affects the percentage of the dataset that can be combined (i.e., unique words with the same key are combined locally to a single KV pair) before and after shuffling in ReduceByKey. In other words, a larger number of unique words means that fewer KV pairs can be combined (i.e., data can be reduced in size) before and after shuffling in ReduceByKey. For example, a dataset of 4 billion words with 72 unique words (4B72) can exhibit up to a 99% combinability rate before the shuffling, whereas a dataset of 4 billion words and 42 million unique words (4B42M) results in only a 25% combinability rate.

3. Performance Analysis

Environment Setting: We measure power usage over time, total energy, and runtime of our miniapplications using two generations of HPC architectures. The first system is a fat node with 68 cores and four hardware threads per core (272 total) the Intel Xeon Phi 7250 Knights Landing (KNL) with a thermal design point (TDP) of 215 watts. The second system is a fat node with 72 cores and four hardware threads per core (288 total) Intel Xeon Phi 7295 Knights Mill (KNM) architecture with the same TDP of 215 watts. We use PAPI to measure the impact of the data management across cores on power. PAPI's event *PACKAGE* allows us to monitor the energy consumption of the entire CPU socket. The PAPI event DRAM_ENERGY monitors the energy consumption of the off-package memory (DDR4). When not otherwise specified, we measure average energy on the core subsystem, including MCDRAM, memory controller, and 2-D mesh interconnect, and on the DDR4 memory at the 100 ms sample rate, a standard rate recommended by the PAPI developers. Finer-grained rates are also assessed. We generate synthetic data that fully fits into the Mimir's system memory, avoiding disk I/O during our tests. We repeat each test three times, with the first of each run shown on graphs to integrate cold-cache effects. We observed a modest variability (less than 2%) across the three runs.

Impact of MapReduce Stages and Architectures: To quantify the impact of the Map+Aggregate versus Convert+Reduce components of the MapReduce job and of different architectures, we measure the power metrics for (a) the combined map operations and aggregate phases of the shuffle stage and (b) GroupByKey (i.e., the complete Map+Shuffle+Reduce workflow) on KNL and KNM using datasets of 4 billion words and 68 unique words on KNL and 72 unique words on KNM, respectively. The number of unique words allows us to fully use the cores available on each HPC architecture. The words are distributed by cutting the dataset in chunks, with each chunk having interleaving sequences of the different words (e.g., at a smaller scale with four unique words "a," "b," "c," and "d," each chuck contains a sequence of "abcdabacdabcd...").

Figures 1a and 1b present power measurement for Map+Aggregate on KNL and KNM, respectively. Figures 1c and 1d present power measurement for GroupByKey on KNL and KNM, respectively. In each figure, we draw three pairs of curves for three power caps: 215, 140, and 120 watts. For each pair, the higher curve is the processor power, and the lower is the DRAM power. We observe that for all the tests, none of our runs exceed 160 watts during runtime (75 watts below system TDP). To study performance impacts, we intentionally impose caps of 140 and 120 watts, which are lower than the miniapplication's max power but higher than the minimum core power usage when idle. We also observe the same patterns for the power usage on both KNL and KNM



(a) KNL Map+Aggregate (68 workds)

(b) KNM Map+Aggregate (72 words)

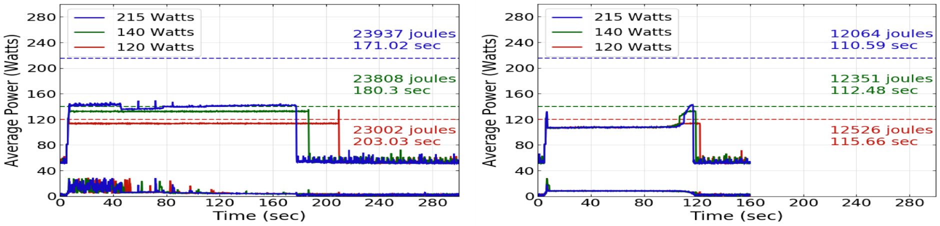
(c) KNL GroupByKey (68 words)

(d) KNM GroupByKey (72 words)

Figure 1. Average power for Map+Aggregate and GroupByKey on KNL and KNM.

architectures; however, tests on KNM have a lower runtime, especially at power limits that are below the normal power usage of the benchmark. This is due to the KNM's higher number of cores. Because of the similar patterns and fact that KNM is the more recent chip, we focus on KNM in the rest of the paper. More important, we observe additional 59–64% runtime associated with the convert phase and reduce operations. The DRAM power forms only a small portion of total power use, making up only between 4% and 15% of the combined processor and DRAM power. Memory power consumption increases during the map and aggregate stages of GroupByKey (and in the Map+Aggregate miniapplication). For Map+Aggregate and GroupByKey, when limiting power to 120 watts, we observe an increase in runtime ranging from 5% to 33%. Comparing Map+Aggregate with GroupByKey allows us to quantify the high runtime and energy costs of the reduce stage. The reduce stage adds 333%–355% more runtime and 335%–359% more energy to Map+Aggregate.

Impact of Combiner Optimizations: MapReduce frameworks have been substantially optimized by switching from the traditional Map+Shuffle+Reduce workflow in GroupByKey to the enhanced workflow based on combiner optimizations in ReduceByKey , for which before, both shuffling and reduce operations, unique words with the same key are combined locally. In general, combiner optimizations can be used for those MapReduce applications that are both associative and commutative (e.g., WordCount), supporting merging KVs with the same key and still providing the correct outcomes. In combiner phases, KV pairs with the same values are combined locally on each node that has just performed the map operation (preshuffling). Once the shuffling of KV pairs is completed and chunks of KV pairs are assigned to processes based on some MapReduce-specific hash function, KV pairs from different processes with the same keys are again combined locally before the reduce operation is performed (postshuffling). From an implementation point of view, in Mimir, the preshuffling and postshuffling processes are executed with combiner callbacks. A first combiner callback is applied before the MPI



(a) GroupByKey – without optimizations

(b) ReduceByKey – with optimizations

Figure 2. Impact of combiner optimizations on power usage in a dataset with combinability rate of 99.99%.

Table 1. Datasets used to quantify the impact of different levels of combinability (combinability rate).

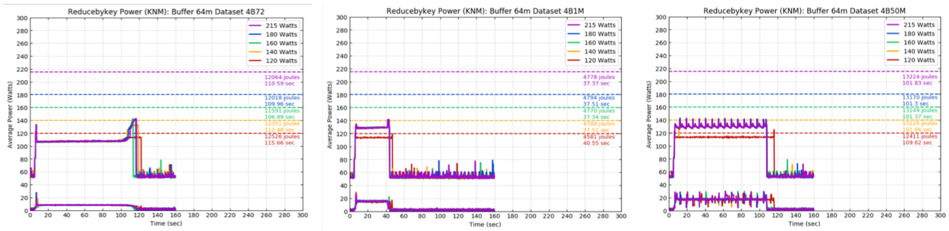
Dataset	Total Words	Unique Words	Combinability Rate
4B72	4B	72	>99%
4B1M	4B	1,111,104	98%
4B50M	4B	49,999,968	10%

communication stage, reducing the communication size, and a second combiner callback is applied after the MPI communication stage, reducing the memory size to store KVs. Figures 2a and 2b show results without combiner optimizations (i.e., GroupByKey) and with combiner optimizations (i.e., ReduceByKey) for a dataset that can exhibit a combinability rate of 99.99% (i.e., the number of KV pairs can be reduced by combining those with the same keys to 0.01% of the initial number because of the very high number of repeated key). Comparing GroupByKey with ReduceByKey allows us to quantify the cost of moving data between processes. Combining KVs before shuffling in ReduceByKey substantially reduces the runtime of the application (up to 46% less runtime) and the power usage over time (up to 11,800 joules saved, or a 50% reduction). One important observation that emerges from this comparison is the intrinsic power cap (without the need of PAPI's cap) that ReduceByKey exhibits for a highly combinable dataset (in Figure 2b). The power usage of the three executions (with 240, 140, and 120 power cap) are all substantially below the defined power caps. In the next two sections we further study the reasons for the implicit capping, by looking into the impacts of the data combinability rate and the MPI buffer size for the shuffling.

Impact of Data Combinability: A KV pair is combined with an existing KV if it is a duplicate of the other KV in the same map process (i.e., the combiner callback combines the new KV with the existing KV). Given a dataset and its number of unique words, the fraction of a dataset that is combinable (called combinability rate) can be calculated as follows:

$$Rate = \frac{N_w - (N_p * N_u)}{N_w}, \tag{1}$$

where N_w is the number of total words, N_u is the number of unique words, and N_p is the number of processors. Table 1 presents the datasets used in this paper with the total number of words (N_w), the number of unique words (N_u), and the combinability rate.



(a) 4B72

(b) 4B1M

(c) 4B50M

Figure 3. Power usage with combiner optimizations for datasets with different combinability rates.

Figures 3a– 3c show the power and time metrics for selected datasets in Table 1, with increasing numbers of unique words and decreasing combinability rates. In the progression of the nine figures, we observe how, when zooming into the combiner performance for datasets with different numbers of unique words (and thus different combinability rates), both the power usage over time and runtime vary around a "sweet spot" region (i.e., a region of minimum runtime and energy). This sweet spot region is around 5,000 unique words for our 4 billion word dataset. For smaller numbers of unique words and high combinability rates, we encounter latency associated with the inability to fill MPI communication buffers and the intrinsic strategy of Mimir to less frequently initiate the shuffling. For larger numbers of unique words, the memory and processor power usage begin oscillating. This oscillatory pattern is due to (1) a lower capability of the combiner optimizations to reduce data chunks in size, (2) a larger number of KV pairs locally managed by each map process in its assigned chunk, and (3) the process's frequent memory accesses to swap parts of the larger chunk in and out of memory.

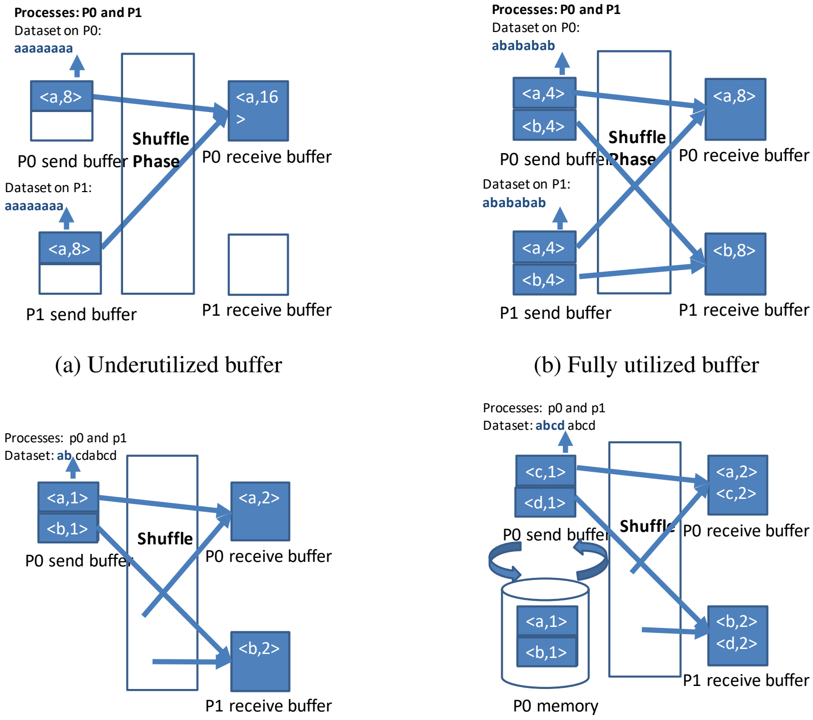
Impact of Communication Buffering: In the tests above, we used a default send buffer size in Mimir of 64 MB. Each map process uses an MPI send buffer to store the KV pairs in groups based on a receiving reduce process. The assignment of a group to a specific process is based on a hash function in Mimir. All the buffers are sent to the reduce processes with an MPI_AllToAllv call when one of the map processes has its send buffer full. In this section, we consider different communication buffer sizes in Mimir (i.e., different sizes triggering the exchange of data among processes in the aggregate phase of the MapReduce workflow) and different data features (i.e., number of unique words in our 4 billion word dataset) to study the impact of Mimir's setting on power usage over time.

Three possible scenarios can be observed. In the first scenario, the send buffer size of each process is partially underutilized because the different unique KV pairs are not filling it. Note that we are using the combiner optimizations to reduce the use of the buffer. This is the case for our dataset of 4 billion words with only 72 unique words used to build the large dataset. Figure 4a shows an example for a small case study of two processes, each one with a data chunk of 8 "a" datasets. The send buffer is divided into two equal-sized partitions, where two is the number of processes executing our MapReduce application. Mimir's default setting temporarily suspends the computation stage and switches to the shuffling when a partition in the send buffer is full. The fact that there are not sufficient unique words to trigger the shuffling results in the execution of map operations and combiner callbacks in an almost-sequential execution. The implicit power cap is the ultimate consequence of a set of cores that are not pushed to their max workload potential. We observe the same phenomena with our datasets and 72 unique words. Figures 5a and 5d show the power usage over time when 72 unique words make up the entire 4 billion words dataset. Figures 5a refers to a scenario in which the send buffer is 32 MB; Figure 5d refers to a scenario in which the send buffer is 64 MB. The buffer that is twice as large results in a larger execution time (almost twice larger), indicating that the larger the buffer,the longer Mimir postpones triggering the aggregate phase in the shuffling. At the same time, the low power usage does not require any intervention with PAPI's power capping.

In the second scenario, the buffer size of each process is fully used: the different unique KV pairs are filling it while the map operations are performed. Figure 4b shows an example for a small case study of two processes, each one with a chunk "abababab" words (where "a" and "b" are the unique words). The send buffer fills regularly, and the aggregate phase is triggered and interleaves with the map operations, resulting in the performance sweet spot. Figures 5e and 5b show the power usage when 5K unique words make up the entire 4 billion word dataset but two different send buffer sizes (i.e., 32 MB and 128 MB). The sweet spot occurs for the same number of unique words, independently from the buffer size. Moreover, the power usage is no longer implicitly defined but is explicitly set up by the PAPI power cap. As with the previous tests, lower power cap results in larger runtime. The performance degradation is within tolerable range.

In the third scenario, the buffer size of each process is overutilized: the different unique KV pairs do not fit in the single partitions of the send buffers. Figure 4c shows an example for a small case study of two processes, each one with the chunk "abacdabcd" (where "a," "b," "c," and "d" are the unique words). At time t, the send buffer fills with the "a" and "b" words, Mimir does not trigger the shuffling but continues the map operations. The buffer has to be copied to the memory. When new map operations on "a" and "b" words are performed, segments of the buffer have to be retrieved, as show in Figure 4d for a hypothetic time $t+1$. We observe the same phenomena with datasets with large number of unique words (e.g., 42 million in Figures 5f and 5c). These two figures show the oscillatory behavior of both CPU and memory power usage over time when moving the buffer through the memory hierarchy (i.e., cache to memory and back). The larger the buffer, the larger the waving pattern observed and, once again, the larger the runtime.

Overhead of Monitoring Frequency: The last aspect we study in this paper is the overhead associated with measuring power usage with PAPI. We consider three sample granularities to collect the power measurements: 100-millisecond sample rate (the default setting), 50-millisecond sample rate, and 5-millisecond sample rate. Values measured by PAPI are energy consumption over the interval of 100 ms, 50 ms, and 5 ms, respectively. The smaller the interval (and thus the more samples collected), the more the overhead. At the same time, the smaller the interval, the more accurate the power values (i.e., less biased by outliers within the sample interval). We measure the power usage for two critical cases: the case in which the system is intrinsically capping the power (e.g., with only 72 unique words in Figures 6a, 6b, and 6c) and the case in which the system forwards parts of the send buffer down the memory hierarchy (e.g., with 42 mission unique words in Figures 6d, 6e, and 6f). The figures show us that the overhead associated with PAPI is marginal and that the average values at 100 ms are sufficiently close to the



(a) Underutilized buffer

(b) Fully utilized buffer

(c) Overutilized buffer (time t)

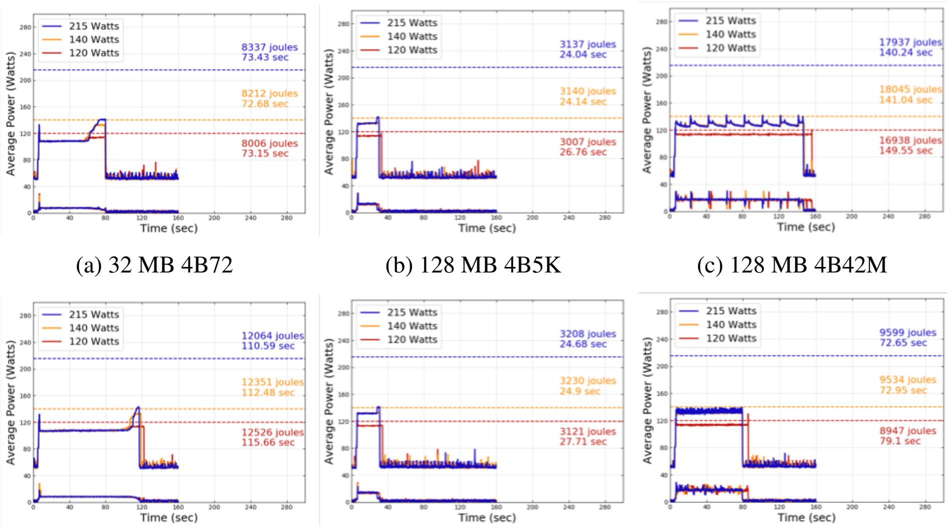
(d) Overutilized buffer (time t+1)

Figure 4. Small scale examples of underutilized, fully utilized, and overutilized send buffers in the map + aggregate phases of ReduceByKey.

values at 5 ms to conclude that power usage does not exhibit variability within the larger interval and there are no outliers within the 100 ms interval.

4. Lessons Learned

We have observed the following from our performance analysis. *First,* combiner optimizations offer significant power performance benefits for our miniapplications. We observe up to a 46% reduction in runtime and up to a 50% reduction in energy consumption when comparing results between GroupByKey and the combiner-optimized ReduceByKey on the same dataset. These results demonstrate the high power performance cost of data management on HPC systems, given the significant performance gain yielded by reducing data management via the combiner optimization. *Second,* he unoptimized reduce stage in the GroupByKey miniapplications has high runtime and energy costs. Comparing GroupByKey with the Map+Aggregate miniapplication, which stops before the reduce stage, shows costs of 333% to 355% more runtime and 335% to 359% more energy consumption when adding the reduce stage to the execution. *Third,* the Map+Aggregate and GroupByKey miniapplications suffer an increase of 5% to 33% more runtime when run under a 120-watt power cap. When running ReduceByKey, with



(a) 32 MB 4B72

(b) 128 MB 4B5K

(c) 128 MB 4B42M

(d) 64 MB 4B72

(e) 32 MB 4B5K

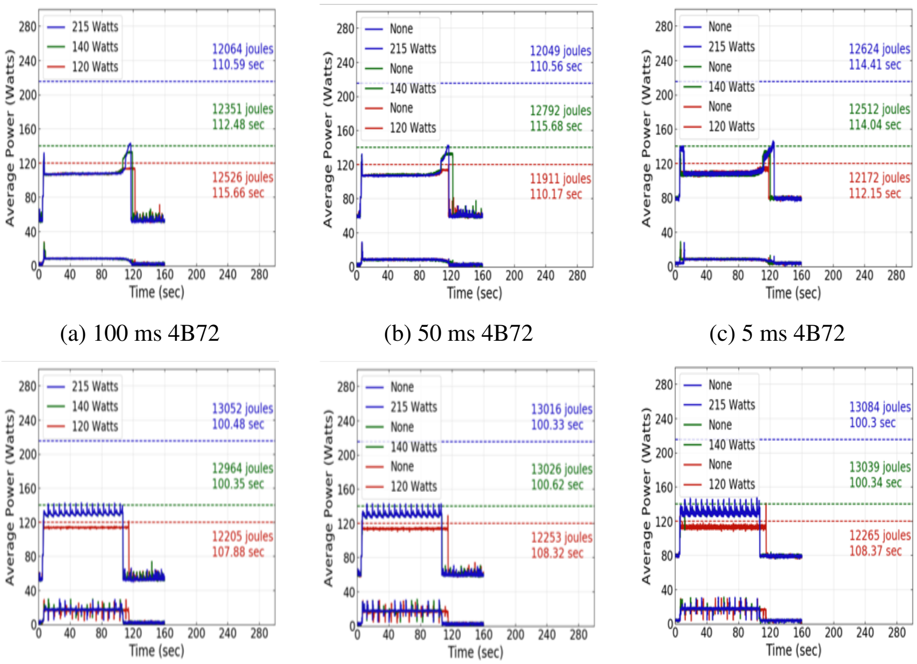
(f) 16 MB 4B42M

Figure 5. Power usage over time and runtime exhibiting underutilized buffers for 4 billion word datasets.

its combiner optimizations, over the same highly combinable dataset, we observe an implicit power cap, below 120 watts. This arises from the very high input dataset combinability and large communication buffer size. *Fourth,* the power performance of the combiner optimizations varies according to the combinability rate of the input dataset around a "sweet spot" region of minimum runtime and energy consumption. This sweet spot is at a combinability rate near 99%, or about 5,000 unique words for a dataset of 4 billion total words. At small numbers of unique words and high combinability rates, performance degrades because of the inability to fill the MPI communication buffers. At large numbers of unique words and low combinability rates, performance degrades because of the increased number of KVs that each process must manage. *Fifth,* the size of the Mimir communication buffer used is significant to the power performance of the combiner optimizations in the ReduceByKey miniapp. Performance degrades when the chosen buffer size is either over- or underutilized, depending on the combinability of the input data. *Last,* the overhead associated with PAPI measurement is marginal, as demonstrated by the lack of impact of decreasing the rate of sampling from 100 to 5 ms. Further, the 100 ms rate is sufficiently fine-grained, because decreasing the sample rate did not lead to greater variability in power usage over time. All these observations can serve as rules of thumb for effective data management using MapReduce over MPI programming languages on HPC architectures. Work in progress is integrating these lessons learned into Mimir, making the framework power-aware (i.e., able to leverage implicit power capping while still controlling power usage in the background, idle to the user).

5. Conclusion

In this paper, we quantitatively measure the impact of data management across cores on power usage for a set of MapReduce-based miniapps that are data intensive on two



(a) 100 ms 4B72

(b) 50 ms 4B72

(c) 5 ms 4B72

(d) 100 ms 4B42M

(e) 50 ms 4B42M

(f) 5 ms 4B42M

Figure 6. Power usage over time with different sample rates of 100 ms, 50 ms, and 5 ms, for a case study with intrinsically power capping (e.g., with only 72 unique words in Figures 6a, 6b, and 6c) and for a case study forwarding parts of the send buffer to memory during the map operations (e.g., with 42 million unique words in Figures 6d, 6e, and 6f).

many-core systems: KNL and KNM. Among our observations, we notice how combiner optimizations lead to up to a 46% reduction in runtime and a 50% reduction in energy usage, without the need for a power cap.

Our future work includes (1) understanding how far our observations are from a general principle relating power cap and performance; (2) studying ways of reducing data movement other than the combiner used in this paper; and (3) understanding how the settings of the underlying MapReduce framework can be tuned during runtime to extend the "sweet spot" regions (i.e., regions of minimum runtime and power usage).

6. Acknowledgments

This work was supported by NSF CCF 1841758.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] —, "Apahce Hadoop," `http://hadoop.apache.org/`.

[3] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *HotCloud*, vol. 10, p. 10, 2010.

[5] X. Lu, F. Liang, B. Wang, L. Zha, and Z. Xu, "DataMPI: Extending MPI to Hadoop-like big data computing," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS)*, 2014.

[6] N. Chaimov, A. Malony, S. Canon, C. Iancu, K. Z. Ibrahim, and J. Srinivasan, "Scaling Spark on HPC systems," in *25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2016, pp. 97–110.

[7] X. Yang, N. Liu, B. Feng, X.-H. Sun, and S. Zhou, "PortHadoop: Support direct HPC data processing in Hadoop," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 223–232.

[8] Y. Wang, R. Goldstone, W. Yu, and T. Wang, "Characterization and optimization of memory-resident MapReduce on HPC systems," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS)*, 2014, pp. 799–808.

[9] —, "IBM BG/Q Architecture," `https://www.alcf.anl.gov/files/IBM_BGQ_Architecture_0.pdf`.

[10] T. Gao, Y. Guo, B. Zhang, P. Cicotti, Y. Lu, P. Balaji, and M. Taufer, "Mimir: Memory-efficient and scalable MapReduce for large supercomputing systems," in *Proceedings of the 31th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017.

[11] T. Gao, Y. Guo, B. Zhang, P. Cicotti, Y. Lu, P. Balaji, and M. Taufer, "On the power of combiner optimizations in MapReduce over MPI workflows." in *Proceedings of the IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, 2018.

[12] T. Gao, Y. Guo, B. Zhang, P. Cicotti, Y. Lu, P. Balaji, and M. Taufer, "Memory-Efficient and Skew-Tolerant MapReduce over MPI for Supercomputing Systems" in *IEEE Transactions on Parallel and Distributed Systems (IEEE TPDS)*, 2019.

[13] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with PAPI-C," pp. 157–173, 2010.

[14] A. Haidar and et al., "Investigating power capping toward energy-efficient scientific applications," *Concurrency Computat Pract Exper.*, 2018.

[15] A. Haidar, H. Jagode, A. YarKhan, P. Vaccaro, S. Tomov, and J. Dongarra, "Power-aware computing: Measurement, control, and performance analysis for intel xeon phi," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2017, pp. 1–7.

[16] F. Ahmad, S. Lee, M. Thottethodi, and T. N. Vijaykumar, "PUMA: Purdue MapReduce Benchmarks Suite," *Technical Report 437, Purdue University*, 2012.

[17] —, "Mimir: MapReduce over MPI," `https://github.com/TauferLab/Mimir-dev.git`.