Execution Measurements of SparkBLAST and CloudBLAST

Supplementary Information for SparkBLAST: Scalable BLAST processing using in-memory operations

Marcelo R. de Castro Catherine dos Santos Tostes
Alberto M. R. Dávila Hermes Senger Fabrício A. B. Silva

June 2017

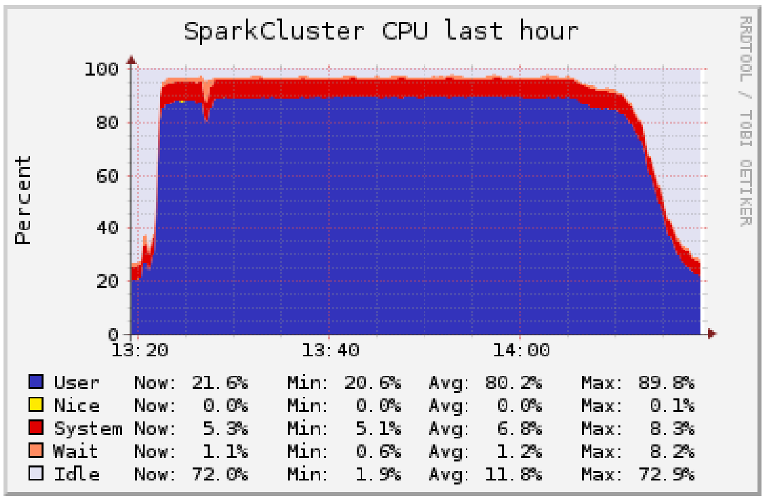
1 Supplementary Data on Experiments

In this supplementary document, we present performance data collected during the execution of Experiment 2 on the Microsoft Azure Platform. All nodes were placed in the same location (East-North US). The cluster is composed of two A4 instances (8 cores and 14GB memory) configured as master nodes and 64 A3 (4 cores and 7GB memory) instances configured as computing nodes. For the measurements presented in this text, both SparkBLAST and CloudBLAST executed queries on the Buz.fasta (805 MB) dataset. We used Galglia [Massie et al. 2004], a scalable distributed monitoring system, to collect the performance data that are presented in this supplementary document.

2 CPU utilization

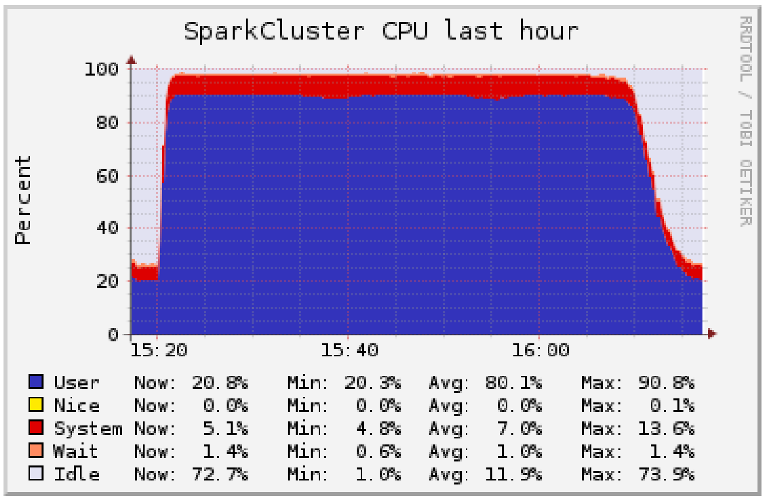
The cluster average CPU utilization is presented in Figures S1 and S2. As illustrated, in both cases CPU utilization reaches near 97-98% after the initialization of the application. CPU utilization did not reach 100% because the master nodes are taken into account and they have low CPU utilization. Individual workers for both systems reach 100% CPU utiilization as depicted in Figure S3 and Figure S4.

As reported in our paper, the execution of CloudBLAST takes a sightly longer time than the SparkBLAST execution. There are some possible reasons for the extra overhead caused by Hadoop. First, Hadoop is much slower than Spark in task initialization [Shi et al. 2015]. Second, there is an overhead caused by the pipe and I/O operations implemented by Hadoop [Ding et al. 2011]. Hadoop uses Linux pipes to redirect *standard input* and *standard output* of BLAST to connect it to the Mapreduce environment. Pipes are inter process communication mechanisms that involve the creation of buffers between two communication processes. The producer process copies data into the buffer, while the consumer process collects and removes data items from the buffer. Both communicating processes have to synchronize. Lastly, the use of data type casting and unbuffered Java I/O operations, which write data to be accesed by external executables (e.g. BLAST) and read data from them, may also lead to performance overhead.



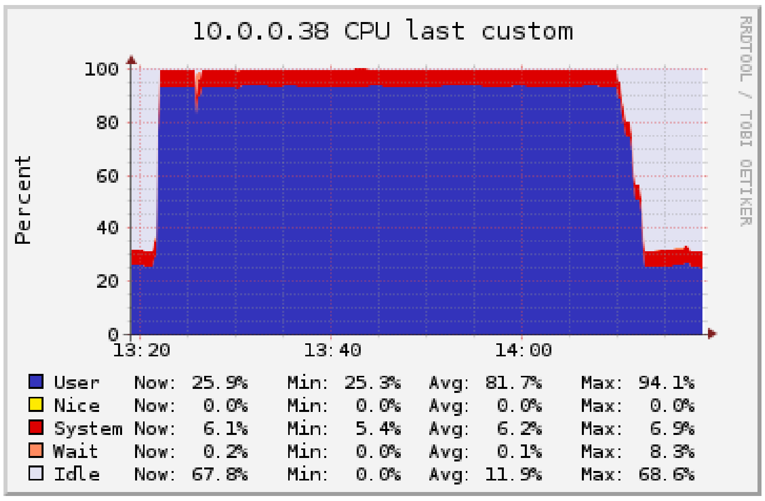
🟦 User	Now: 21.6%	Min: 20.6%	Avg: 80.2%	Max: 89.8%
🟨 Nice	Now: 0.0%	Min: 0.0%	Avg: 0.0%	Max: 0.1%
🟥 System	Now: 5.3%	Min: 5.1%	Avg: 6.8%	Max: 8.3%
🟧 Wait	Now: 1.1%	Min: 0.6%	Avg: 1.2%	Max: 8.2%
⬜ Idle	Now: 72.0%	Min: 1.9%	Avg: 11.8%	Max: 72.9%

Figure S1: CPU utilization for the SparkBLAST execution.



🟦	User	Now: 20.8%	Min: 20.3%	Avg: 80.1%	Max: 90.8%
🟨	Nice	Now: 0.0%	Min: 0.0%	Avg: 0.0%	Max: 0.1%
🟥	System	Now: 5.1%	Min: 4.8%	Avg: 7.0%	Max: 13.6%
🟧	Wait	Now: 1.4%	Min: 0.6%	Avg: 1.0%	Max: 1.4%
⬜	Idle	Now: 72.7%	Min: 1.0%	Avg: 11.9%	Max: 73.9%

Figure S2: CPU utilization for the CloudBLAST execution.

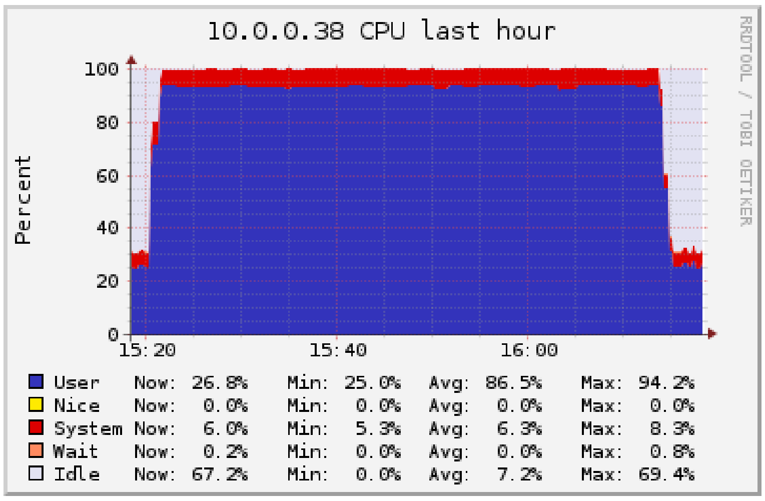


🟦	User	Now: 25.9%	Min: 25.3%	Avg: 81.7%	Max: 94.1%
🟨	Nice	Now: 0.0%	Min: 0.0%	Avg: 0.0%	Max: 0.0%
🟥	System	Now: 6.1%	Min: 5.4%	Avg: 6.2%	Max: 6.9%
🟧	Wait	Now: 0.2%	Min: 0.0%	Avg: 0.1%	Max: 8.3%
⬜	Idle	Now: 67.8%	Min: 0.0%	Avg: 11.9%	Max: 68.6%

Figure S3: CPU utilization for one worker node running SparkBLAST.

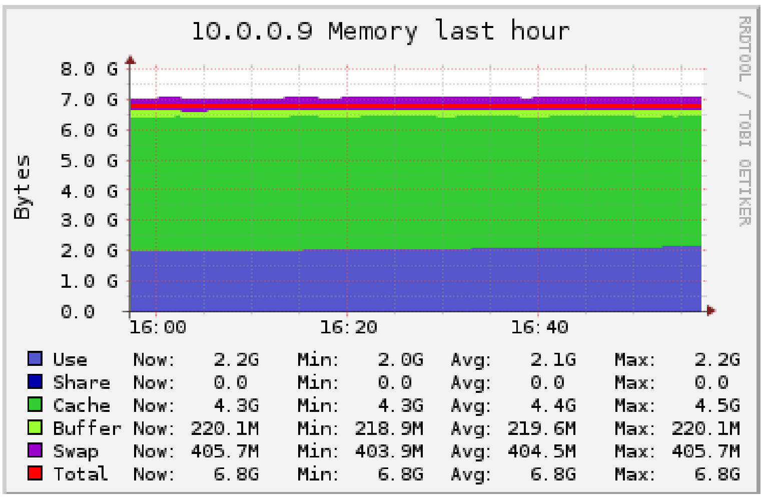
3 Memory utilization

Another cause for superior performance of SparkBLAST is memory management. Spark implements the Resilient Distributed Datasets (RDDs), which implement in-memory data structures used to cache intermediate data across a set of nodes. The effect can be observed in Figures S5 and S6. Hadoop needs to use more memory than Spark, while Spark can maintain a larger cache and less swap to execute.



```
■ User   Now:  26.8%   Min:  25.0%   Avg:  86.5%   Max:  94.2%
■ Nice   Now:   0.0%   Min:   0.0%   Avg:   0.0%   Max:   0.0%
■ System Now:   6.0%   Min:   5.3%   Avg:   6.3%   Max:   8.3%
■ Wait   Now:   0.2%   Min:   0.0%   Avg:   0.0%   Max:   0.8%
□ Idle   Now:  67.2%   Min:   0.0%   Avg:   7.2%   Max:  69.4%
```

Figure S4: CPU utilization for one worker node running CloudBLAST.

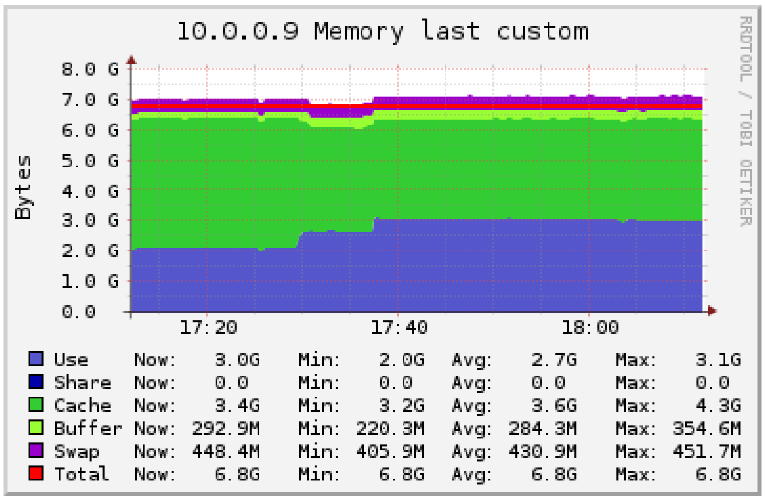


```
■ Use    Now:   2.2G    Min:   2.0G    Avg:   2.1G    Max:   2.2G
■ Share  Now:   0.0     Min:   0.0     Avg:   0.0     Max:   0.0
■ Cache  Now:   4.3G    Min:   4.3G    Avg:   4.4G    Max:   4.5G
■ Buffer Now: 220.1M    Min: 218.9M    Avg: 219.6M    Max: 220.1M
■ Swap   Now: 405.7M    Min: 403.9M    Avg: 404.5M    Max: 405.7M
■ Total  Now:   6.8G    Min:   6.8G    Avg:   6.8G    Max:   6.8G
```

Figure S5: Memory utilization for SparkBLAST

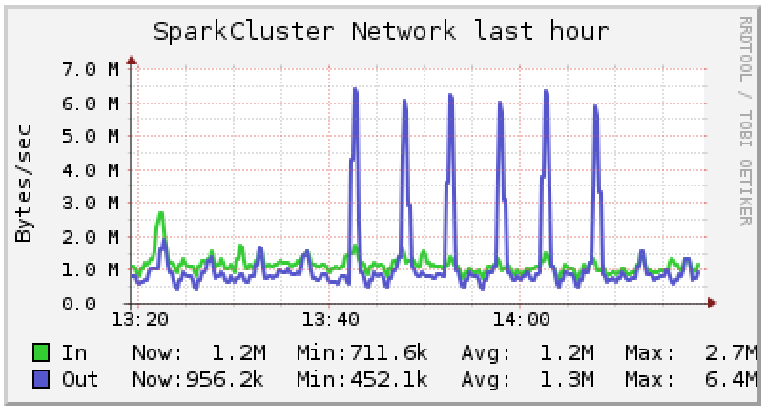
4 Network traffic

The network traffic produced by SparkBLAST and CloudBLAST are presented in Figure S7 and Figure S8, respectively. The total amount of data transmitted over the network by SparkBLAST during its execution is 3.76 GB, while CloudBLAST's traffic reaches 8.92 GB. CloudBLAST experienced a traffic peak in the beginning of the execution because it transfers the data from the blob service to the local disk.



🟦 Use	Now:	3.0G	Min:	2.0G	Avg:	2.7G	Max:	3.1G
🟦 Share	Now:	0.0	Min:	0.0	Avg:	0.0	Max:	0.0
🟩 Cache	Now:	3.4G	Min:	3.2G	Avg:	3.6G	Max:	4.3G
🟩 Buffer	Now:	292.9M	Min:	220.3M	Avg:	284.3M	Max:	354.6M
🟪 Swap	Now:	448.4M	Min:	405.9M	Avg:	430.9M	Max:	451.7M
🟥 Total	Now:	6.8G	Min:	6.8G	Avg:	6.8G	Max:	6.8G

Figure S6: Memory utilization for CloudBLAST.



🟩 In	Now:	1.2M	Min:	711.6k	Avg:	1.2M	Max:	2.7M
🟦 Out	Now:	956.2k	Min:	452.1k	Avg:	1.3M	Max:	6.4M

Figure S7: Network traffic produced by SparkBLAST during its execution.

5 Final Remarks

Although Hadoop and Spark are designed to support the execution of distributed applications on large clusters, their design and implementation lead to different performance. Compared to Hadoop, Spark has demonstrated better performance for the execution of BLAST applications on a cloud platform. Possible reasons for the superior performance of Spark come from its low overhead in task management, better memory utilization and less network traffic. Also, the underlying mechanisms used to implement Hadoop streaming also lead to extra overhead.

References

[Ding et al. 2011] Ding, M., Zheng, L., Lu, Y., Li, L., Guo, S., and Guo, M. (2011). More convenient more overhead: the performance evaluation of hadoop streaming. In *Proceedings of the 2011 ACM Symposium on Research in Applied Computation*, pages 307–313. ACM.

[Massie et al. 2004] Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840.

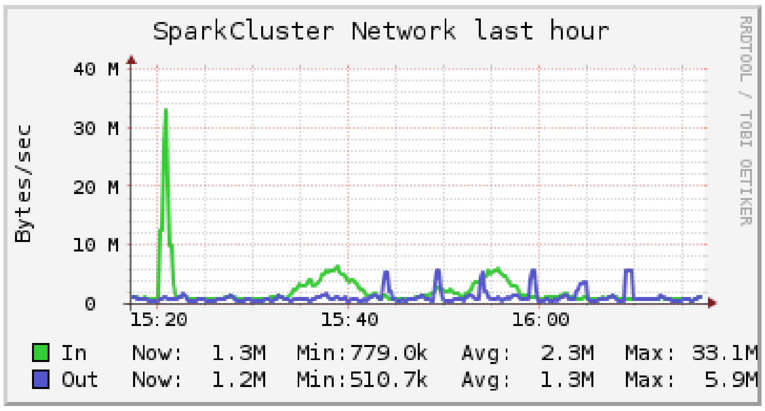


Figure S8: Network traffic produced by CloudBLAST during its execution.

[Shi et al. 2015] Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., and Özcan, F. (2015). Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proc. VLDB Endow.*, 8(13):2110–2121.