Benchmarking X-Stream and Graphchi

Laurent Bindschaedler
LABOS, EPFL
laurent.bindschaedler@epfl.ch

Amitabha Roy
LABOS, EPFL
amitabha.roy@epfl.ch

1 Introduction

A number of graph processing systems have been proposed that operates using secondary storage on a single machine. In the interests of furthering research we (LABOS) and others (Graphchi, CMU) have released the source code of X-Stream [1] and Graphchi [2] as examples of such systems in source code form. The initial set of comparisons between X-Stream and Graphchi in the SOSP 2013 paper were limited to a one type of SSD. We extend those benchmarks in this short report by considering a variety of storage devices. In addition, our own experience with benchmarking both these systems has shown that it can often be tricky for newcomers to property set up and benchmark them. Therefore we present detailed notes on our benchmark runs to ensure that others can replicate them and avoid subtle errors.

2 Environments

The detailed environment for our benchmarking is as follows.

2.1 Software

We benchmark X-Stream (version 0.9). This is version 0.8 with a minor bugfix that incorporates changes to the way Linux does direct I/O with more recent kernel versions.

We benchmarked Graphchi (https://github.com/GraphChi/graphchi-cpp), checked out on June 2, 2014.

2.2 Machines

- LABOS: This is the same machine as used in our SOSP 2013 paper with some minor changes. AMD Opteron (6272, 2.1Ghz), dual-socket, 32-core system with 32GB memory. It has two Intel 200 GB PCI Express SSDs (Intel) arranged in a software RAID-0 configuration and three 3TB 7200 RPM magnetic disks arranged in a software RAID-0 configuration.
- Cambridge: Use of this machine is courtesy Eiko Yoneki at the University of Cambridge. This is an AMD Opteron(6344, 2.6Ghz), dual-socket, 24-core system with 256 GB of memory. It has a number of SSDs that were used for benchmarking: A Samsung 840, OCZ Vertex, two Samsung 840s arranged in a software RAID-0 configuration and finally a 1TB 7200 RPM magnetic disk. The system runs stock Ubuntu 12.04.

Before proceeding further we remark that the vendor names have been mentioned for completeness. The benchmark results should not be taken to reflect the relative capabilities of the devices (especially the SSDs). Extracting good performance from X-Stream (and we presume, Graphchi) requires tuning them and this document discusses out of the box benchmarking only.

3 Setup

We benchmarked weakly connected components and pagerank on both the systems for the widely used snapshot of the Twitter followers graph [3]. This data is directed and therefore we convert it to an undirected graph (by including a reverse edge for every directed edge) in order to compute connected components. We executed both Graphchi and X-Stream using 8 GB of memory (in order to ensure that the graph is not cached). The configuration of the two were as follows:

3.1 X-Stream

```
benchmark_driver -p 8 -a -b pagerank -g twitter_rv --physical_memory 8589934592
benchmark_driver -p 8 -a -b cc -g twitter_rv-undirected --physical_memory 8589934592
```

On all the machines we used 8 cores for the benchmarks. Increased core counts showed little benefit to performance since the bottleneck in all cases was secondary storage.

3.2 Graphchi

We setup graphchi.conf to contain the following values (recommended for use on an 8GB system).

```
# GraphChi configuration.
# Commandline parameters override values in the configuration file.
execthreads=16
loadthreads = 4
niothreads = 2

# Good for 8gigs
membudget_mb = 2000
cachesize_mb = 1000

# I/O settings
io.blocksize = 1048576
mmap = 0  # Use mmaped files where applicable


# Comma-delimited list of metrics output reporters.
# Can be "console", "file" or "html"
metrics.reporter = console,file,html
metrics.reporter.filename = graphchi_metrics.txt
metrics.reporter.htmlfile = graphchi_metrics.html
```

The command lines used were:

```
bin/example_apps/connectedcomponents file twitter_rv-und.tsv filetype edgelist
/bin/example_apps/pagerank_functional mode sync file twitter_rv.tsv filetype edgelist niters 5
```

An easy to make benchmarking error is to assume that these settings are sufficient to make Graphchi run using 8GB of memory on machines with more RAM (ours have 64GB and 256GB respectively). Graphchi uses read, write and mmap to access data from disk. This leaves the operating system free to cache data in its pagecache which (on Linux) grows to fill all available memory. Therefore simply running with these settings for Graphchi would cause the graph to be cached entirely in RAM with no I/O and easily outperforming any system that is using secondary storage (such as X-Stream with the settings above).

Our solution was to run a background program that allocated and then mlocked all RAM except 8GB and then went to sleep. This utility program (`utils/block_mem.cpp`) is available in the 0.9 release of X-Stream. It ensures that exactly 8GB of memory including the pagecache are available to Graphchi. Note that X-Stream does direct I/O and therefore does not use the pagecache.

Another important point to keep in mind is the input to the two. X-Stream takes as input an unordered edge list in a binary file format (records consisting of a 4 byte source, 4 byte destination and 4 byte random weight). Graphchi takes as input a comma separated tsv file with the source and destination. Our starting point for the measurement is when these files are available and X-Stream and Graphchi are ready to run. We do (as a matter of courtesy) provide simple python scripts to convert tsv files to the X-Stream format in our release. These single threaded python are not meant to be benchmarked (or even intended to be fast) so we do not report any numbers for them.

4 Results

In the case of X-Stream, we measure runtime as the reported wall time which is an output of the form:

```
<INFO 0x1b3f740> CORE::TIME::WALL 1835.5 seconds
```

This is the time taken from program start to finish. X-Stream does not do any pre-processing as it streams unordered edge lists.

In the case of Graphchi, we report from the output the time taken for sharding (preprocessing), which is a line of the form:

```
execute_sharding: 962.695 s
```

and the time taken for actual execution of the algorithm, which is a line of the form:

```
runtime: 4138.51 s
```

The results for the runs are shown in the Table 1 for connected components and in Table 2 for pagerank. We report the average of five runs in seconds and include the 99% confidence intervals around the average.

5 Summary

The results from our experiments largely confirms the observations in our SOSP paper [1] that X-Stream outperforms Graphchi even when the pre-processing time in Graphchi is excluded, a comparison that is unfair to X-Stream as it starts from an unordered edge list that Graphchi needs to sort and shard. A few points however stand out. First, RAID with the Samsung 840 does not deliver the scaling performance that we would have expected with both Graphchi and X-Stream. This is possibly due to configuration issues with the RAID and the need for tuning of the IO block size in both Graphchi and X-Stream. The second

System	Graphchi (shard)	Graphchi (run)	X-Stream
LABOS			
Intel SSDs	965.401 ± 6.174	4101.284 ± 49.696	1171.766 ± 3.993
Disk	1087.612 ± 17.225	5801.214 ± 106.371	1821.172 ± 17.598
Cambridge			
Samsung 840	748.569 ± 35.408	3933.532 ± 732.388	1797.546 ± 108.369
2xSamsung 840	674.993 ± 117.829	3372.794 ± 181.577	1445.540 ± 188.704
OCZ Vertex	735.388 ± 109.804	4190.728 ± 166.380	2069.136 ± 23.720
Disk	956.619 ± 25.546	5936.218 ± 2798.894	5338.932 ± 559.120

Table 1: Results for connected components

System	Graphchi (shard)	Graphchi (run)	X-Stream
LABOS			
Intel SSDs	486 ± 6.762	908.966 ± 16.667	417.213 ± 3.037
Disk	591.848 ± 19.885	1507 ± 13.656	616.795 ± 2.271
Cambridge			
Samsung 840	389.569 ± 41.879	943.246 ± 19.754	588.613 ± 5.259
2xSamsung 840	375.729 ± 35.975	811.359 ± 23.706	443.396 ± 40.446
OCZ Vertex	423.104 ± 5.218	1079.138 ± 20.600	843.023 ± 276.625
Disk	590.584 ± 55.165	1879 ± 93.368	1613.174 ± 106.151

Table 2: Results for pagerank

point is that the gap between Graphchi and X-Streams appears to be quite narrow with the single hard disk on the Cambridge system. The reason why the runtime of Graphchi is generally larger than X-Stream is that the time required to create in-memory shards by resorting is dominant. For example, with pagerank on the Samsung 50% of the Graphchi runtime is spent creating in-memory shards. However, with a slower drive the I/O dominates in both Graphchi and X-Stream. For example with pagerank only 30% of the Graphchi runtime is spent in creating memory shards. Due to the slower I/O device, both are held to this lowest common denominator.

We hope that this short report illustrates how to benchmark both Graphchi and X-Stream. For X-Stream we are extremely keen to hear back from people using and benchmarking it and would be more than glad to help explain output and investigate any anomalies that might be observed.

References

[1] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. X-stream: Edge-centric graph processing using streaming partitions. In *Proceedings of the ACM symposium on Operating Systems Principles*, pages 472–488. ACM, 2013.

[2] Aapo Kyrola and Guy Blelloch. Graphchi: Large-scale graph computation on just a PC. In *Proceedings of the conference on Operating Systems Design and Implementation*. USENIX Association, 2012.

[3] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *Proceedings of the International conference on World Wide Web*, pages 591–600. ACM, 2010.