Big Data Benchmarking with High Level Tasks

A Spectrum of Needs

It is important to recognize that with big-data benchmarking, both very low-level and very high-level work-loads are needed for testing. Examples of low-level work-loads include file creation rate, unreplicated and replicated read and write rates, task creation rates and program invocation times. All of these measurements are extremely useful for the platform engineer tasked with tuning a particular aspect of system performance. Mid-level Hadoop oriented work-loads include things like tera-sort, GridMix and PigMix. All of these help compare and tune systems across similar architectures. They do little, however, when comparing implementations of widely differing architectures and they cannot provide much information about realistic multi-job or multi-tenant situations.

For example, even something as simple as a detailed breakdown of terasort task timing can be used to tune most aspects of a Hadoop cluster's isolated job performance. This performance tuning applies more or less even for jobs with very different characteristics from the terasort because the different phases of map-reduce program execution do not exhibit much interaction. That lack of interaction, in turn, means that tuning each phase separately for terasort will generally impact a wide range of programs. Of course, an aggregation job that has enormous input, little shuffle and almost no output will benefit from input bandwidth tuning, but it will generally not be adversely affected by tuning the shuffle or output phases.

The fact that terasort details tell us how to tune isolated map-reduce programs, however, tells us very little about how something as closely related as a map-reduce-reduce architecture might run a sequence of jobs and it tells us next to nothing about how a real-time system such as Storm would be able to handle a large number of records.

Low and mid-level benchmarks also give very little information about the cost of shimming between differing kinds of systems. For instance, in an aggregation system that combines a requirement for live aggregate measurements with a requirement to retain those aggregates reliably for several years, no amount of low or mid-level benchmarks would highlight the performance penalty of copying data from the real-time part of the system to the long-term storage system. Some systems such as standard Hadoop require such a copy since their storage layer cannot support real-time requirements reliably while other systems much be able to host the real-time portion of the system directly without any copy at all. Higher level tasks are required that incorporate such integration point costs.

Looking backwards, while database benchmarks have helped enormously in tuning relational databases, few database oriented benchmarks have proven very useful in assessing the value of Hadoop systems because the change in semantics. Hadoop's strengths relate to the ability to stream data and apply schemas at read-time. Such options were never even considered a realistic possibility when the various TPC

benchmarks were designed. Conversely, the cost of transactions was a major consideration in historical database benchmarks while with map-reduce, the major progress has been due exactly to the extent that transactions can so often be made irrelevant.

These considerations indicate the need for higher-level benchmarks in addition to the more traditional low and mid-level tests. These high-level benchmarks need to be oriented around realistic tasks but should allow considerable flexibility in detailed implementation to allow the virtues of alternative architectures to be exploited. Different frameworks each have different algorithmic sweet spots so fixing the algorithmic choice is often tantamount to pre-determining the outcome of the test, usually to the detriment of the benchmarks value.

Lessons from the Past

The history of benchmarking in numerical linear algebra provides some interesting contrasts that should help us design big data benchmarks. The SPEC suites are quite prescriptive and thus provide little scope for comparing software improvements other than limited kinds of compiler improvements. The result of this rigidity is that the SPEC benchmarks need to be updated frequently as workloads change and desired software mix changes. In contrast, the LINPACK benchmarks allow for considerable innovation in the underlying implementations. In addition, the LINPACK benchmarks are inherently scalable in that you can run them with a scale parameter that indicates how large a problem you would like work on. Related eigensolver and FFT benchmarks provide similar capabilities. The result of being less prescriptive and scalable is that the LINPACK benchmarks are still useful roughly 30 years after they were first proposed.

The lessons to be learned from this history are that at least some big data benchmarks need to be high-level (to allow introduction of new algorithms and frameworks) and scalable. The requirement for scalability implies that the benchmarks use synthetic data, which must, of course, represent realistic challenges, and neither be susceptible to excessive compression or shortcuts nor so uniformly random that compression has no impact.

Possible Tasks

There are a number of interesting tasks that might provide the basis of a reasonable set of benchmarks. These tasks all exhibit a few fundamental properties:

- They use realistic inputs derived from resampling real data or by generating random data from realistic distributions. This allows the benchmarks to be freely distributed.
- They allow tests to be scaled to very large sizes. This is important to allow the “big” in big data to stay big.
- They relate directly to real problems which makes the implications of performance to be readily understandable by a non-technical audience.

- They are complex enough that they cannot be expressed as a simple map-reduce or a single query. This is important to highlight systems that allow optimization across program unit boundaries.
- They produce large outputs or require large intermediate files as opposed to simple aggregation jobs.

In addition, some of the tasks such as the real-time/long-term aggregation task exercise a requirement for data to cross system boundaries.

Not stated here, but implicit in any big data system is a requirement that performance

Real-time+long-term aggregation

The task here is to accept data containing symbols and to provide real time counts of the number times each kind of symbol has been seen as well as the number of unique symbols in a non-overlapping time window. The time windows should vary in size from 10 seconds to 1 year. The system should accumulate counts and make the counts available within 1 second of a record arriving, but provide full resolution counts from any time period over the last 3 years of data.

The input symbols should be drawn from a long-tailed distribution Pittman-Yar distribution and it should be possible to build realistic historical data faster than real-time.

Scaling can be done by increasing the input data rate and by varying the parameters of the input distribution.

The point of this task is to test the ability to process data that is large in two axes, rate and size.

Co-occurrence counting

The task here is to accept two kinds of transaction files. Each transaction would contain a user id, a time and a token of type A or B. The task is to join the transaction files into temporally ordered user histories and then to count the number of times tokens of type A occur within a time window before tokens of type B subject to a constraint that abnormally common tokens can be down-sampled to a specified maximum number of occurrences and abnormally high frequency user histories can be down-sampled to have no more than a specified maximum number of type A or type B tokens within any given time window. Cooccurrences within the window that are not down-sampled would be counted and processed using a statistical score such as a log-likelihood ratio test. The output would be a summary table with tokens of type A as key and a list of no more than a maximum related type B tokens.

The performance metric would consist of the time required to process data of different sizes.

The goal of this task is to emulate certain kinds of collaborative filtering workload with realistic constraints and offers many opportunities for platform optimization such as the use of map-reduce-reduce or map-side coalescence.

Inverted index creation

The task here is to index text-like synthetic documents at a high aggregate rate for use by a text retrieval engine. The size of the documents should be tunable and the vocabulary should be generated by a process that emulates real text. Changing the total number of documents can scale the task or a real-time variant with a variable number of documents per second can be posed. In the batch version, the total index time would be the performance metric while in the real-time version, the performance metric would be delay until the documents are available for searching.

Insider trading or account take-over detection

The task here is to take data that consists of synthetic equity trades to find suspiciously timed trades. A variant on this task would be to find common points of compromise for synthetic accounts that demonstrate a suspicion score.

For the insider-trading scenario, the input would be synthetic equity pricing and trade data that occasionally demonstrates large changes in price. The goal would be to find sets of trades that beneficially anticipate these shifts that have common characteristics.

For the account compromise variant, the input would be synthetic transaction data that has an associated fraud score. The goal would be to find sets of accounts that have a moderately elevated fraud score that also have a common preceding transaction merchant.

For both cases, the performance metric would be the time required to process different scales of problem.

Genome sequence assembly against reference genome

The task here is take a large number of short genome sequences which exhibit various mutations relative to a reference genome and produce a reconstructed genome. The input data could be based on synthetic mutations of the reference genome, but the reference genomes should be real. The problem can be scaled by allowing many reference genomes (the multi-species variant) or by allowing many different mutated forms of the same reference genome (the multiple mutation variant).

The performance metric would be the time required for the reconstruction.