Large-scale Incremental Processing Using Distributed Transactions and Notifications

Daniel Peng and Frank Dabek

Presented by Nick Radcliffe

What is Percolator?

- Percolator is a system for incrementally processing updates to a large data set.
- Percolator has been used to produce Google's websearch index since April, 2010.

What is Percolator?

- Percolator updates the index **incrementally** as new documents are crawled.
- “Incremental” is a big deal because it avoids redoing work that has already been done.

Benefits of MapReduce

- It's easy to maintain data invariants when MapReduce is used to update the search index.
- A series of MapReduces must be run to process the document repository, but these MapReduce phases are serialized.

Benefits of MapReduce

- For example, the indexing system must perform link inversion, as well as determine the PageRank of each page.
- However, inverted links are only written to the highest-PageRank URL.
- When the indexing system writes inverted links to the current highest-PageRank URL, the PageRank cannot change because the MapReduce phase that computes it has already finished.

Why not MapReduce?

- Say you update the search index after recrawling some small portion of the web.
- If MapReduce is used to update the index, then it must process the *entire* repository, not just the new documents.
- A series of 100 MapReduces must be used to process the repository.

Why not simply Bigtable?

- Updating the search index can be done much more efficiently if *random accesses* of the repository are possible.
- If the computation does not have strong consistency requirements, then a distributed storage system such as Bigtable is sufficient.

Why not simply Bigtable?

- Bigtable can scale to the size of the Google repository, but...
- Bigtable does not give programmers tools to maintain data invariants in the face of concurrent updates to the index.
- In particular, Bigtable does not provide multirow transactions, which are needed for incrementally updating Google's index.

Summary of why Percolator is so great

- Random accesses to the document repository while maintaining data invariants, which allows...
- Incremental updating of the search index, so...
- No global scans of the entire repository, therefore...
- Average age of documents in Google search results decreased by 50%.

Tradeoffs

- Percolator trades efficient use of resources for scalability.
- Caffeine (the Percolator-based indexing system) uses twice as many resources as the previous system to process the same crawl rate.
- Roughly 30 times more CPU per transactions than a standard DBMS.

Bigtable distributed storage system

- Bigtable presents a multi-dimensional sorted map to users.
- Keys are (row, column, time stamp).
- Bigtable handles petabytes of data and runs reliably on large numbers of (possibly unreliable) machines.

Bigtable distributed storage system

- Bigtable provides lookup and update operations on each row, and
- Bigtable row transactions enable atomic read-modify-write operations on individual rows (but *not multiple rows*).

Bigtable distributed storage system

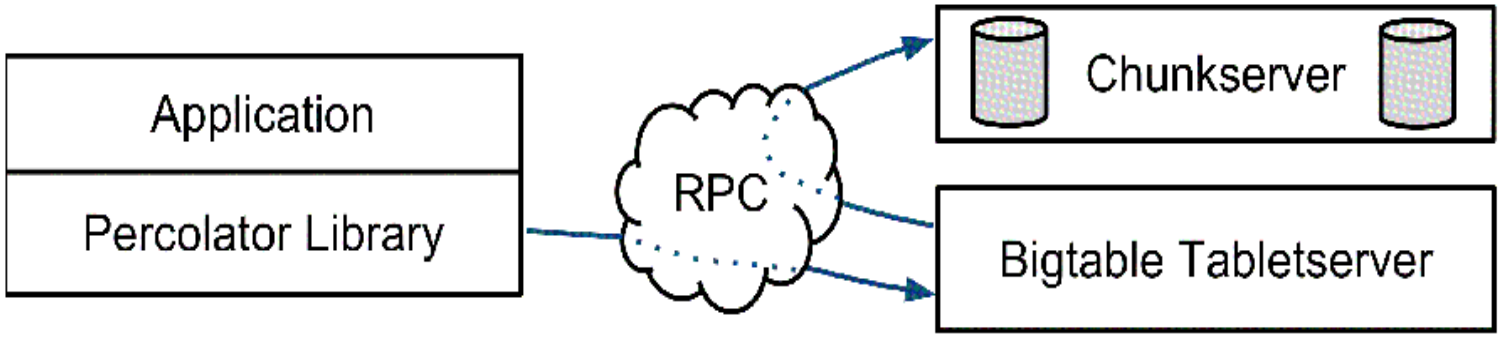
- A running Bigtable consists of a collection of tablet servers, each of which is responsible for serving several tablets
- (tablet = contiguous region of key space).
- The operation of the tablet servers is coordinated by a master, i.e., the master might direct a server to load/unload a tablet.

Bigtable distributed storage system

- A tablet is stored as a collection of read-only files in the Google SSTable (Sorted String Table) format.
- The SSTables are stored in GFS, and Bigtable relies on GFS to preserve data in the event of disk loss.

Bigtable demonstration

Overview of Percolator design



- Percolator is built on top of Bigtable.
- A percolator system consists of three binaries that run on every machine in the cluster: a Percolator worker, a Bigtable tablet server, and a GFS chunkserver.

Overview of Percolator design

- Data is organized into Bigtable rows and columns, with Percolator metadata stored alongside in special columns.
- The Percolator library largely consists of Bigtable operations wrapped in Percolator-specific computation.
- Percolator adds **multirow transactions** and **observers** to Bigtable.

Overview of Percolator design

- An *observer* is like an event-handler that is invoked whenever a user-specified column changes.
- Percolator applications are structured as a series of observers.
- Each observer completes a task and creates more work for “downstream” observers by writing to the table.

Percolator transactions

- Percolator provides cross-row, cross-table transactions with ACID snap-shot isolation semantics.
- It is possible to incrementally process data without the benefit of strong transactions.
- Transactions make it more tractable for the user to reason about the state of the system.

Percolator transactions

- A Get() operation first checks for a lock in the timestamp range [0, start timestamp], which is the range of timestamps visible in the transaction’s snapshot.
- If no conflicting lock is found, Get() reads the latest write record in that timestamp range and returns the data item corresponding to that write record.

Percolator transactions

- If a client fails while a transaction is being committed, locks will be left behind.
- Deadlock is possible if the locks are not cleaned up.
- When a transaction A encounters a conflicting lock left behind by transaction B, A may determine that B has failed and erase its locks.

Percolator transactions: code snippet

```
bool UpdateDocument(Document doc) {
  Transaction t(&cluster);
  t.Set(doc.url(), "contents", "document", doc.contents());
  int hash = Hash(doc.contents());

  // dups table maps hash → canonical URL
  string canonical;
  if (!t.Get(hash, "canonical-url", "dups", &canonical)) {
    // No canonical yet; write myself in
    t.Set(hash, "canonical-url", "dups", doc.url());
  } // else this document already exists, ignore new copy
  return t.Commit();
}
```

Timestamps

- The timestamp oracle is a server that hands out timestamps in strictly increasing order.
- The transaction protocol uses strictly increasing timestamps to guarantee that Get() returns all committed writes before the transaction’s start timestamp.

Timestamps

- Since every transaction requires contacting the timestamp oracle twice, this service must scale well.
- Timestamp requests are batched to decrease RPC's and hence increase the scalability of the oracle.

Notifications

- The user writes code (“observers”) to be triggered by changes to the table.
- Each observer registers a function and a set of columns with Percolator.
- Percolator invokes the function after data is written to one of those columns in any row.

Notifications

- Percolator applications are structured as a series of observers.
- Each observer completes a task and creates more work for “downstream” observers by writing to the table.

Notifications

- In Google's indexing system, a MapReduce loads crawled documents into Percolator by running loader transactions.
- These transactions trigger the document processor transaction to index the document (parse, extract links, etc.).

Notifications

- The document processor transaction triggers further transactions, like clustering.
- The clustering transaction triggers transactions to export changed document clusters to the serving system.

Caffeine

- The Percolator-based indexing system (called Caffeine) crawls the same number of documents as the old system.
- The main design goal of Caffeine is a reduction in latency.

Caffeine

- The median document moves through Caffeine over 100x faster than the previous system.
- This latency improvement grows as the system becomes more complex.
- Adding a new clustering phase to Caffeine requires an extra lookup for each document rather an extra scan over the repository.

Caffeine

- For Caffeine, more resources are required to make sure the system keeps up with the input.
- For the old system, no amount of resources can overcome delays introduced by stragglers in an additional pass over the repository.

Document clustering delay

- The authors used a synthetic benchmark that clusters newly crawled documents against a billion document repository to remove duplicates.
- This is similar to the way that Google's indexing pipeline operates.

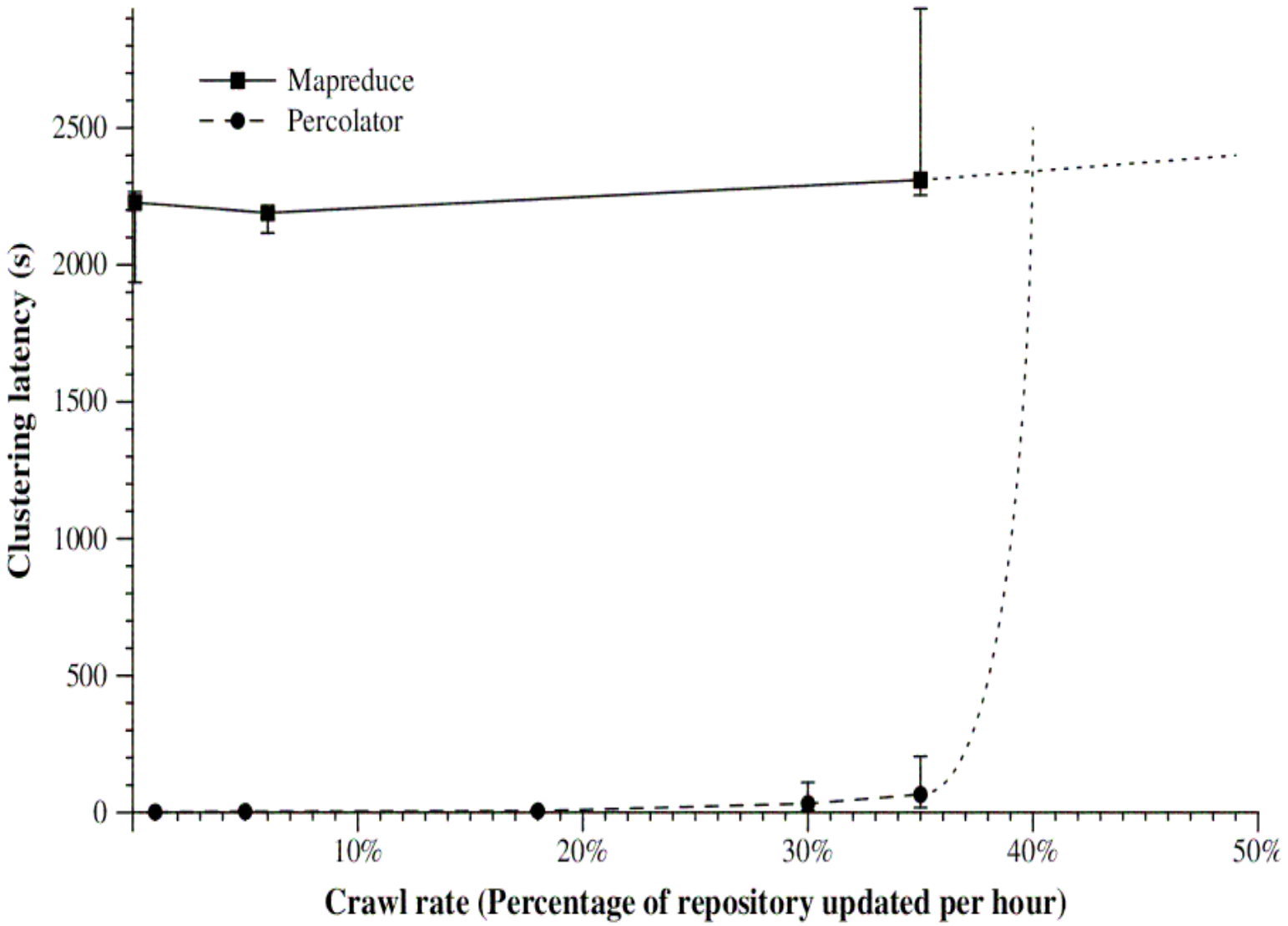
Document clustering delay

- In the Percolator clustering implementation, each crawled document is immediately written to the repository to be clustered by an observer.
- The observer maintains an index table for each clustering key and compares the document against each index to determine if it is a duplicate.

Document clustering delay

- MapReduce implements clustering of continually arriving documents by repeatedly running a sequence of clustering MapReduces.
- The sequence of MapReduces processes the *entire repository* and any crawled documents that accumulated during previous MapReduce phases.

Document clustering delay



Microbenchmarks: Percolator vs Bigtable

- Comparison of Percolator to a “raw” Bigtable.
- Only interested in the relative performance of Bigtable and Percolator.

	Bigtable	Percolator	Relative
Read/s	15513	14590	0.94
Write/s	31003	7232	0.23

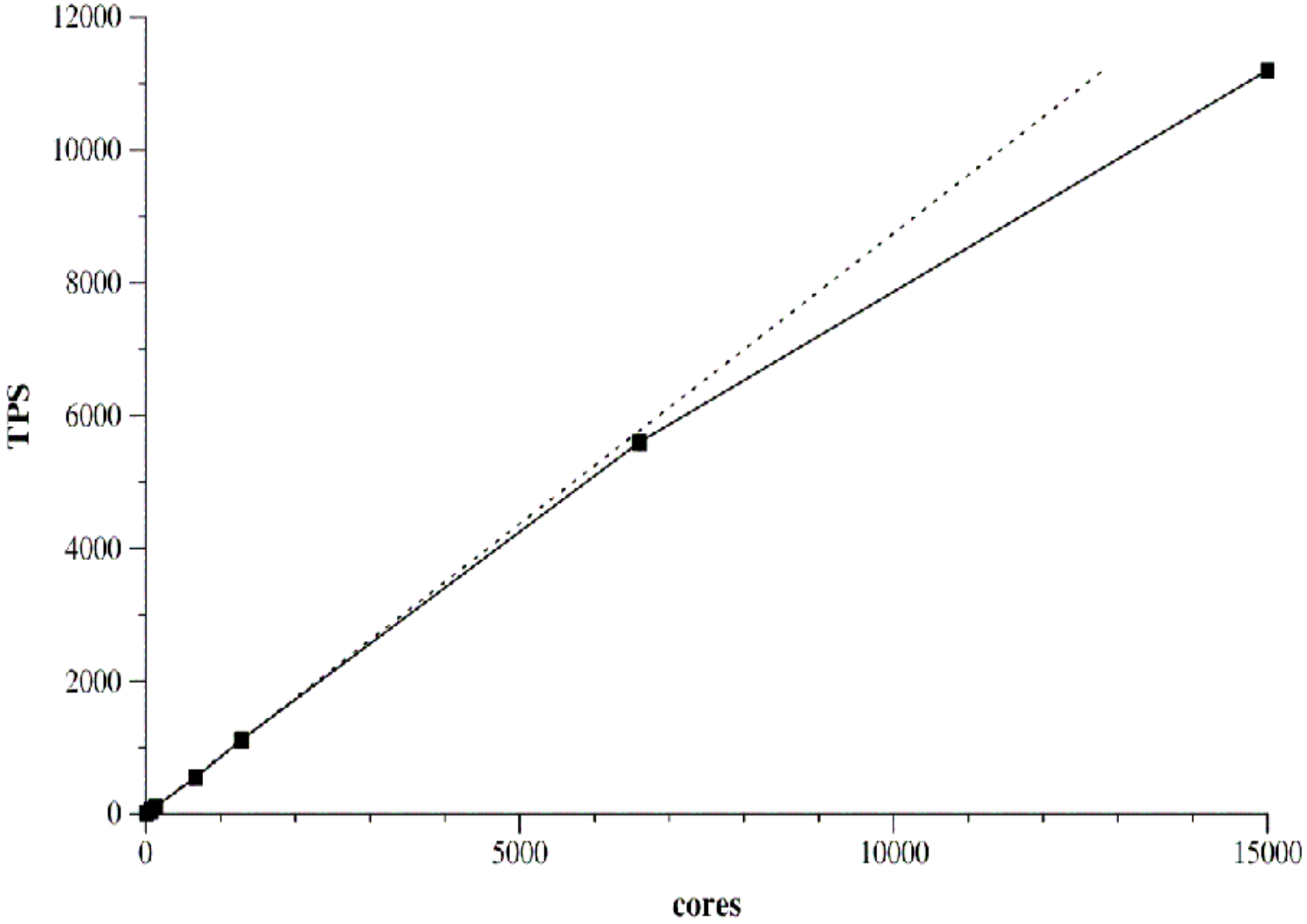
Percolator and TPC-E

- TPC-E simulates a brokerage firm with customers who perform trades, market search, and account inquiries.
- The authors' implementation is a combined customer/market emulator that calls into the Percolator library to perform operations against Bigtable.

Percolator and TPC-E

- TPC-E was used to evaluate Percolator on a more realistic workload.
- A number of Percolator’s tradeoffs conflict with desirable properties of DBMS's that TPC-E was designed for.

Percolator and TPC-E



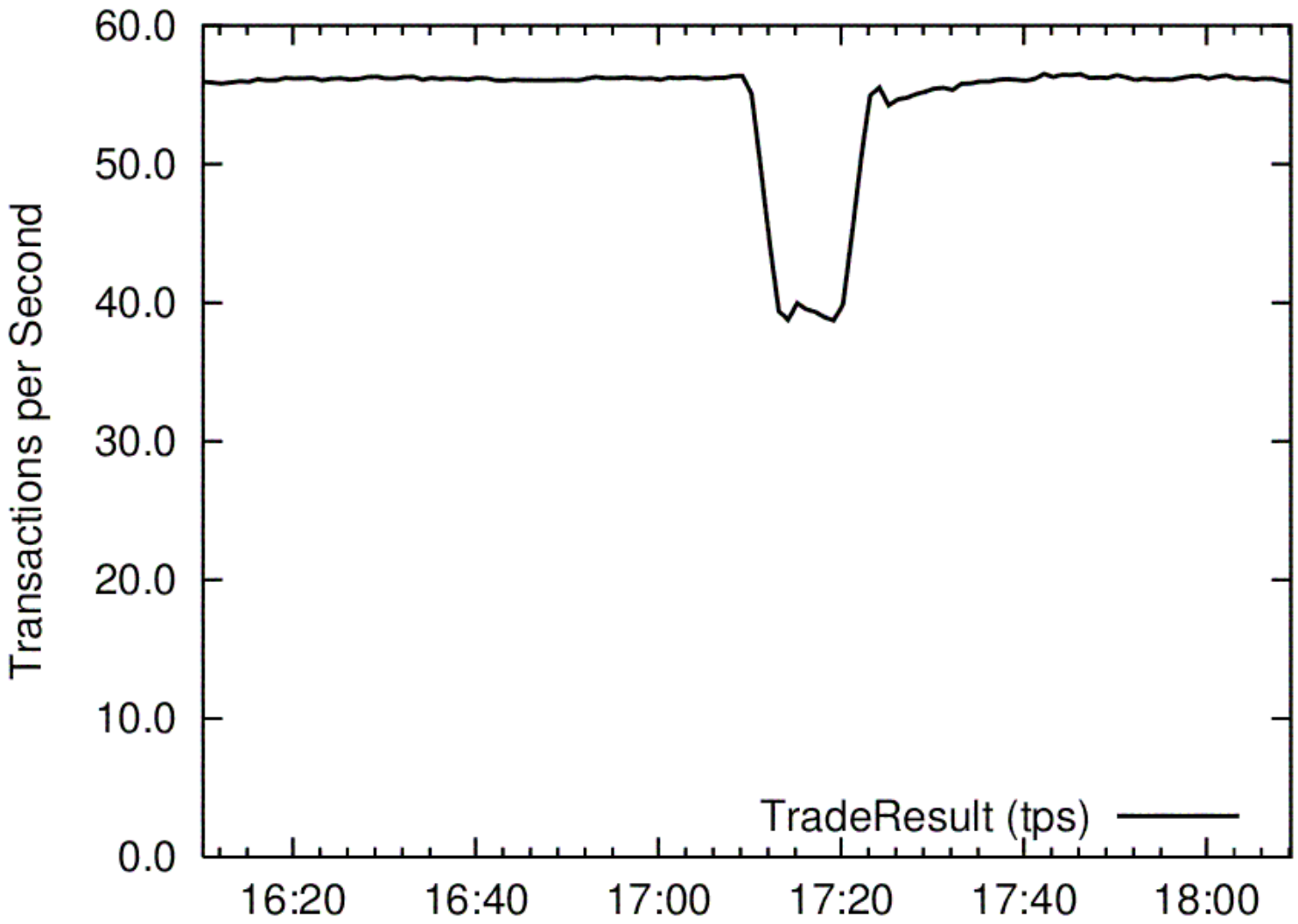
Percolator and TPC-E

- The authors estimate that Percolator uses roughly 30 times more CPU per transaction than the benchmark system.
- However, Percolator is scalable and it's performance on the benchmark (using 15,000 cores) is three times the current record (which used 64 cores).
- 15,000 > 64.

Percolator, TPC-E, and Failures

- To test fault tolerance, the benchmark was run with 15 tablet servers and performance was allowed to stabilize.
- The next plot shows the performance of the system over time.
- The dip in performance at 17:09 corresponds to a third of the tablet servers being (temporarily) killed.

Percolator, TPC-E, and Failures



Questions

- Is the reduction in the average age of documents in Google search results worth the extra resources required by Percolator/Caffeine?
- Can the scalability of Percolator be obtained without the substantial performance overheads?
- (From the authors) How much of an efficiency loss is too much to pay for the ability to add capacity endlessly simply by purchasing more machines?