Think Like a
{Vertex, Column, Parallel Collection}

David Konerding, Google Inc.

Pregel: a system for large-scale graph processing
Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski
SIGMOD’10

Dremel: Interactive Analysis of Web-Scale Datasets
Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis
VLDB’10

FlumeJava: Easy, Efficient data-parallel pipelines
Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, Nathan Weizenbaum
PLDI’10
Google’s data-intensive parallel processing toolbox

MapReduce is already well-known; external implementations are becoming popular in industry and academia.

MR is not designed to handle many kinds of problems, so in the past few years we have developed new toolkits/frameworks for doing data-intensive parallel processing.

Some common situations where we need alternatives:
  • Large graph operations with multiple steps.
  • Interactive tools for data analysts dealing with trillion-row datasets.
  • Pipelines with complex data flow
Think Like a Vertex

**Pregel: a system for large-scale graph processing**
Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski
SIGMOD’10

Most similar existing framework: Parallel Boost Graph
Model of graph computation

- computation on local data (parallelism, !deadlock, !race)
- "batch&push" communication, no "pull" (!latency)
- message sending overlaps with computing
- synchronization barriers (programmability)
class ShortestPathVertex : public Vertex<int, int, int> {
  public:
    virtual void Compute(MessageIterator* messages) {
      int min_dist = IsSource(vertex_id()) ? 0 : INT_MAX;
      for (; !messages->Done(); messages->Next()) {
        min_dist = min(min_dist, messages->Value());
      }
      if (min_dist < GetValue()) {
        *MutableValue() = min_dist;
        OutEdgeIterator iter = GetOutEdgeIterator();
        for (; !iter.Done(); iter.Next()) {
          SendMessageTo(iter.Target(),
            min_dist + iter.GetValue());
        }
      }
      VoteToHalt();
    }
};

vertex value is initialized to INT_MAX
Implementation

**master:**
load graph, compute, checkpoint, restore, save, exit

**workers:**
register, report result of operation

Graph partitioned across workers. Partitions reside in workers' memory
Fault-tolerance

Daly, FGCS '06:

optimal time between checkpoints = \( \sqrt{2 \times C \times M} - C \)

- **C** = [constant]
- **M** = mean time to [Poisson] failure

**load graph**

**superstep 0**

**superstep 1**

**superstep 2**
Usage of Pregel at Google

Easy to program and expressive

- Breadth-first search
- Strongly connected components
- PageRank
- Label propagation algorithms
- Minimum spanning tree
- $\Delta$-stepping parallelization of Dijkstra's SSSP algorithm
- Several kinds of vertex clustering
- Maximum and maximal weight bipartite matchings
- many more!

Used in dozens of projects at Google
Think Like a Column

Dremel: Interactive Analysis of Web-Scale Datasets
Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis
VLDB’10

Most similar external application: Hadoop Pig
Dremel

- Trillion-record, multi-terabyte datasets
- Scales to thousands of nodes
- Interactive speed
- Nested data
- Columnar storage and processing
- *In situ* data access (e.g., GFS, Bigtable)
- Aggregation tree architecture
- Interoperability with Google's data management tools (e.g., MapReduce)
Query processing

- Data model: ProtoBufs (~nested relational)
- Select-project-aggregate (single scan)
  - Most common class of interactive queries
  - Aggregation within-record and cross-record
  - Filtering based on within-record aggregates
- Fault-tolerant execution
- Approximations: count(distinct), top-k
- Joins, temp tables, UDFs/TVFs, etc.
- Limited support for recursive types
Record versus column oriented data

Record-oriented

Column-oriented
Performance Breakdown comparing record reads to column reads

- (a) read + decompress
- (b) assemble records
- (c) parse as objects
- (d) read + decompress
- (e) parse as objects

Graph showing time (sec) vs. number of fields for reading from records and columns.
Mixer tree

- Client
- Query execution tree
- Root server
- Intermediate servers
- Leaf servers (with local storage)
- Storage layer (e.g., GFS)
- Fault tolerance, re-execution
Example: count(*)

- Select A, COUNT(B) FROM T GROUP BY A
  T = {/gfs/1, /gfs/2, ..., /gfs/100000}

- Select A, SUM(c) FROM (R_1_1 UNION ALL R_1_10) GROUP BY A

- Select A, COUNT(B) AS c FROM T_1_1 GROUP BY A
  T_1_1 = {/gfs/1, ..., /gfs/10000}

- Select A, COUNT(B) AS c FROM T_2 GROUP BY A
  T_2 = {/gfs/10001, ..., /gfs/20000}

- Select A, COUNT(B) AS c FROM T_2_1 GROUP BY A
  T_2_1 = {/gfs/1}

- File::PRead()
Widely used inside Google

- Analysis of crawled web documents
- Tracking install data for applications on Android Market
- Crash reporting for Google products
- OCR results from Google Books
- Spam analysis
- Debugging of map tiles on Google Maps
- Tablet migrations in managed Bigtable instances
- Results of tests run on Google's distributed build system
- Disk I/O statistics for hundreds of thousands of disks
- Resource monitoring for jobs run in Google's data centers
- Symbols and dependencies in Google's codebase
Think Like a Parallel Collection

**FlumeJava: Easy, Efficient data-parallel pipelines**
Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, Nathan Weizenbaum
PLDI’10

Most similar external application: Hadoop Cascading, Pipes, Dryad/LINQ
Parallel Collections

- PCollection\(<T>, \ PTable\(<K,V>\):
  (possibly huge) parallel collections
  - parallelDo(DoFn) \(\leftrightarrow\) Map() equivalent
  - groupByKey() \(\leftrightarrow\) Shuffle() equivalent
  - combineValues(CombineFn) \(\leftrightarrow\) Combiner() / Reducer() equivalent
  - flatten(...)
  - readFile(...), writeToFile(...)

- Work with Java data & control structures
  - join(...), count(), top(CompareFn,N), ...

PCollection<String> lines =
    readTextFileCollection("/gfs/data/shakes/hamlet.txt");
PCollection<DocInfo> docInfos =
    readRecordFileCollection("/gfs/webdocinfo/part-*",
      recordsOf(DocInfo.class));
Example: TopWords

```
readTextFile("/gfs/corpus/*.txt")
.parallelDo(new ExtractWordsFn())
.count()
.top(new OrderCountsFn(), 1000)
.parallelDo(new FormatCountFn())
.writeToTextFile("cnts.txt");
```

FlumeJava.run();
Deferred Evaluation &
The Execution Graph

- Primitives, e.g., `parallelDo(...)`, are “lazy”
  - Just append to execution graph
  - Result PCollections are like “futures”
- Other code, e.g., `count()`, is “eager”
  - “Inlined” down to primitives
- `FlumeJava.run()` “demands” evaluation
  - Optimizes, then runs execution graph
Optimizer

• Fuse trees of parallelDo operations into one
  – producer-consumer
  – co-consumers (“siblings”)
  – eliminate now-unused intermediate PCollections
• Form MapReduces
  – pDo + gbk + cv + pDo ➔
    MapShuffleCombineReduce (MSCR)
  – multi-mapper, multi-reducer, multi-output
Initial pipeline
After sinking Flattens and lifting CombineValues
After ParallelDo fusion
After MSCR Fusion
Executor

- Runs each optimized MSCR
  - If small data, runs locally, sequentially
    - develop and test in normal IDE
  - If large data, runs remotely, in parallel
- Handles creating, deleting temp files
- Supports fast re-execution
  - Caches, reuses partial pipeline results
Experience

- Released to Google users in May 2009
  - Now: hundreds of pipelines run by hundreds of users every month
  - Pipelines process gigabytes \(\rightarrow\) petabytes

- Typically, find FlumeJava a lot easier to use than MapReduce
  - Can exert control over optimizer and executor if/when necessary
  - When things go wrong, lower abstraction levels intrude
Think Like a
{Vertex, Column, Parallel Collection}

David Konerding, Google Inc.

**Pregel: a system for large-scale graph processing**
Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski

**Dremel: Interactive Analysis of Web-Scale Datasets**
Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis

**FlumeJava: Easy, Efficient data-parallel pipelines**
Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, Nathan Weizenbaum
Conclusions

- All tools are fault-tolerant by design- failure of individual nodes just slows down completion.
- Work at large scale (trillions of rows, billions of vertices, petabytes of data).
- Used by multiple groups inside Google.
- We expect external developers will implement technologies similar to Pregel, Dremel and FlumeJava within Hadoop.