MapReduce

the silver bullet of scalability(?)

Tobias K Torrissen

What is MapReduce?

- programing model popularized by google
- inspired by map and reduce functions
- large datasets distributed on clusters of computers
- word counting

sweet spot

"Data intensive, time-consuming, serial task, such as a migrations, indexing, crawling etc"

Map

- higher order function
- applies function to a list
- myNewList = myList.map(func)
- [1,2,3].map(x*x) -> [1,4,9]

Reduce

- Family of higher order functions
- AKA: fold, accumulate, compress, inject
- [1,2,3,4].fold(“,”,func) ->[10]

putting it together

putting it together

- input

putting it together

- input
- partitioning

putting it together

- input
- partitioning
- map()

putting it together

- input
- partitioning
- map()

```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");
```

putting it together

- input
- partitioning
  - map()
  - partition

```
map(String input_key, String input_value):
   // input_key: document name
   // input_value: document contents
   for each word w in input_value:
      EmitIntermediate(w, "1");
```

putting it together

- input
- partitioning
- map()
- partition
- reduce()

```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");
```

putting it together

- input
- partitioning
- map()
- partition
- reduce()

```
map(String input_key, String input_value):
   // input_key: document name
   // input_value: document contents
   for each word w in input_value:
      EmitIntermediate(w, "1");
```

```
reduce(String output_key, Iterator intermediate_values):
      // output_key: a word
      // output_values: a list of counts
      int result = 0;

      for each v in intermediate_values:
         result += ParseInt(v);
      Emit(AsString(result));
```

putting it together

- input
- partitioning
- map()
- partition
- reduce()
- output

```
map(String input_key, String input_value):
   // input_key: document name
   // input_value: document contents
   for each word w in input_value:
      EmitIntermediate(w, "1");
```

```
reduce(String output_key, Iterator intermediate_values):
      // output_key: a word
      // output_values: a list of counts
      int result = 0;

      for each v in intermediate_values:
         result += ParseInt(v);
      Emit(AsString(result));
```

pro

- Simple

  1. write a map function
  2. write a reduce function
  3. gogogo!

- “Just run a MapReduce over it”

pro

- Scalability
  - Just add more boxes
- Reliability
  - boxes can die but the run will complete.
- Proven
  - yahoo, google, et.al

con

- Things have to be embarrassingly parallel?
- Not good at re-computations
- If you don´t have a some kind of shared memory or distributed file system you will might have a single point of failure/IO as a bottleneck.

MapReduce vs other grids

- MapReduce is basicly a subset of
- Master/Worker
- MPI

Silver bullet of scalability?

- Well, as always: choose the right tool for the job.
- Consider MapReduce if you have large amounts of data you want to do simple computations on.

Criticism

David DeWitt, Michael Stonebraker:

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all — it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

Implementations

- Hadoop
- GridGain
- Disco
- CouchDB