SciCloud: Adapting Scientific computing problems to Cloud

Pelle Jakovits, Satish Srirama

Distributed Systems Group,
University of Tartu

SciCloud project

Scientific Computing on the Cloud

- Goal is to use existing university resources to setup private clouds
- Enabling students and researchers to perform large scale scientific calculations
- Based on the Eucalyptus cloud computing platform
- Our current experimental platform consists of 8+16+14 = 38 cores
- My role in this project is adapting scientific computing problems to Cloud

Why use Cloud?

- Public clouds promise virtually infinite resources:
  - These resources can be used for HPC needs in Scientific Computing
  - Hardware used is mostly commodity computers
  - Which are bound to fail regularly
- Are there any frameworks that efficiently use such cloud computing resources?
- How to adapt the scientific computing applications to these frameworks?

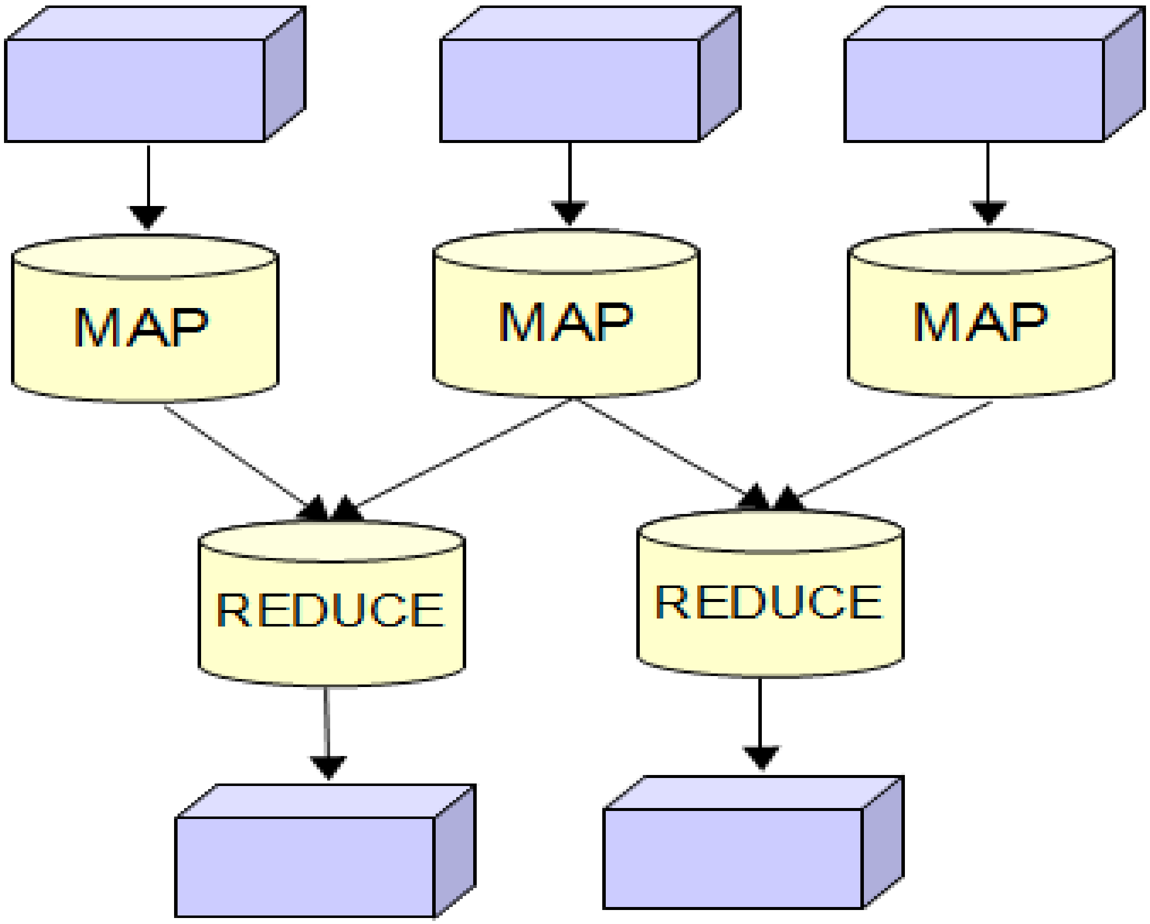
MapReduce framework

- First developed by Google, for huge scale data processing
- Has automatic parallelization
  - Simplifies writing distributed computing applications
  - Allowing to focus on implementing algorithms instead of managing background tasks.
- MapReduce framework:
  - Handles scheduling, communication, synchronisation
  - Has built in fault tolerance
  - Works on top of a distributed file system
  - Moves processes to the data
- However, MapReduce algorithm structure is very strict

MapReduce model

- Input is a list of key and value pairs: <k,v>
- Programmers specify two functions:
  - **map** <k, v> → <k’, v’>*
  - **reduce** <k, v*> → <k’, v’>*
    - All values with the same key are reduced together
- The execution framework can handle everything else
- But often, programmers also specify:
  - **partition** (k’, number of partitions) → partition for k’
    - Often a simple hash of the key, e.g., hash(k’) mod n
    - Divides up key space for parallel reduce operations
  - **combine** <k, v*> → <k’, v’>*
    - Used as an optimization to reduce network traffic

Parallelism in MapReduce



Hadoop MapReduce

- Inspired by Google's proprietary MapReduce framework.
- Developed by Apache software foundation under free licence.
- In constant use by:
  - Yahoo! 36,000 nodes with 100,000 CPUs more than 25 different Hadoop clusters
  - Facebook 1400 8-core nodes
  - Ebay 532 8-core nodes
  - LinkedIn 240 8-core nodes
  - Twitter, IBM, Adobe, AOL, ...

Adapting algorithms to MapReduce

- Implemented different iterative algorithms on Hadoop Mapreduce framework, like:
  - Conjugate Gradient (CG)
  - K-Medoid clustering
    - Partitioning Around Medoids (PAM)
    - Clustering Large Applications (CLARA)
- Results were:
  - Hadoop MapReduce is not well suited for iterative algorithms.

Conjugate Gradient (CG)

- Linear system solver.
- Complex iterative algorithm.
- Matrix and Vector operations used in each iteration must be adapted to MR separately.
- As a result 4 MR jobs executed at every iteration.

Partitioning Around Medoids (PAM)

- Iterative clustering algorithm using medoids.
  - Medoid – the most central element representing the whole cluster.
- Only needs a distance operator for objects to be clustered
- Whole iteration can be expressed as a single MR job.

Problems with Hadoop

- Complex algorithms require iterating over a number of MR jobs in sequence.
- In Hadoop, each MR cycle is a separate program execution.
- As a result:
  - Each time it takes time to start up and finish a job. ~20 sec
  - Every time, the input must be read again from the file system.

Twister MapReduce

- MapReduce for iterative algorithms.
- Long running MapReduce tasks.
- Starting up a job takes ~ 3 sec.
  - Not affected by the number of iterations
- Data can be kept in memory between MR executions.
- No distributed file system provided
- No fault tolerance
- Less stable

Hadoop vs Twister

- CG with Hadoop (Time in seconds)

	500	1000	2000	4000	8000
1	261	327	687	1938	7619
2	259	298	507	1268	4185
4	236	281	360	721	2193
8	251	291	397	563	1246
12	260	281	420	543	949

- CG with Twister (Time in seconds)

	500	1000	2000	4000	8000
1	3	3	3	5	11
2	3	3	3	4	7
4	3	3	3	4	5
8	3	3	3	4	5
12	3	3	3	3	4

Hadoop vs Twister

- PAM with Hadoop (Time in seconds)

	10000	25000	50000	75000	100000
1	1389	1347	2014	3620	6959
2	1133	1697	1826	2011	6130
4	803	782	1156	2562	2563
8	635	627	1513	1084	1851

- PAM with Twister (Time in seconds)

	10000	25000	50000	75000	100000
1	5	21	25	97	205
2	3	10	23	51	93
4	4	8	15	16	92
8	4	5	15	32	38

Conclusions

- Twister is 80 to 500 times faster than Hadoop
- Twister can solve larger problems.
- Twister needs more memory to be effective.
  - The problem must fit into the collective memory of the machines used.
- Twister has much shorter startup time. But still too high for real time applications. (~3 sec)

Further work

- Also interested looking at other MapReduce frameworks like
  - HaLoop
  - Spark
- Bulk Synchronization Parallel model
  - Pregel – Google framework for graph computing
- Come up with a design for our own cloud computing framework.

Thank you!

Any questions?