PDAD
Parallel Data Analysis Diff

Cristina Băsescu

Claudiu Gheorghe

SCPD

Motivation

- IT reached a turning point in its evolution
- We have to think in terms of 'the power of the group'
- Many applications (intrisically) suitable for distributed processing
  - Data processing
  - Anything that needs to scale
- what framework/paradigm/programming language/library is suitable for developing the application?

Goals

- Answering the previous slide question ☺
- Tools
  - Hadoop's MapReduce
  - Hadoop's Pig
  - MPI
- Comparison between them regarding
  - Performance
  - Productivity
  - Scalability
  - Portability
  - Tuning

Applications

- Statistics on large amount of data
- Structured, real, large (tens of GB) input data:

Inria Failure Trace Archive

- Frequent fault reasons
- Most frequent causes of failures depending on job duration
- Frequent end reasons for events
- Number of failures for each geographical location

Approach – a typical JOIN

- Number of failures for each geographical location

  (platform_id, node_id, ...) X (platform_id, node_id, location)

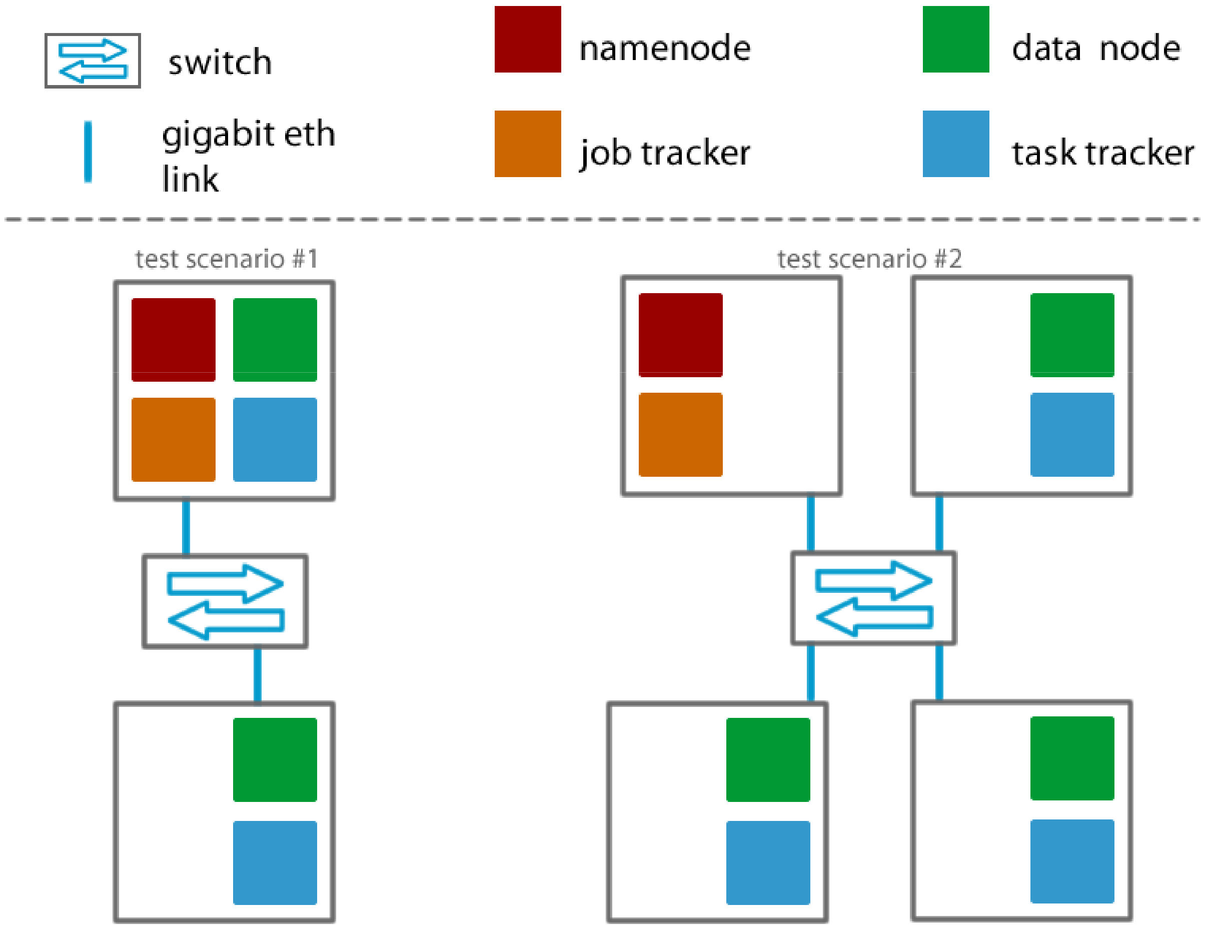
- MapReduce: 2 jobs
  - Mapper reads both files, emits (platform_id;node_id,1) and (platform_id;node_id,location)
  - Combiner emits (platform_id;node_id,x) and (platform_id;node_id,location)
  - Reducer emits multiple (location,y)
  - Mapper – indetity
  - Combiner and Reducer sum up the values, emit (location,z)

Approach – a typical JOIN (2)

- Pig: straight forward join keyword

- MPI: complex master-slave design
    - Master keeps the smaller file in memory
    - Explicit distribution of the other file
    - Slaves computes events
    - queries the master about location corresponding to (platform_id, node_id)

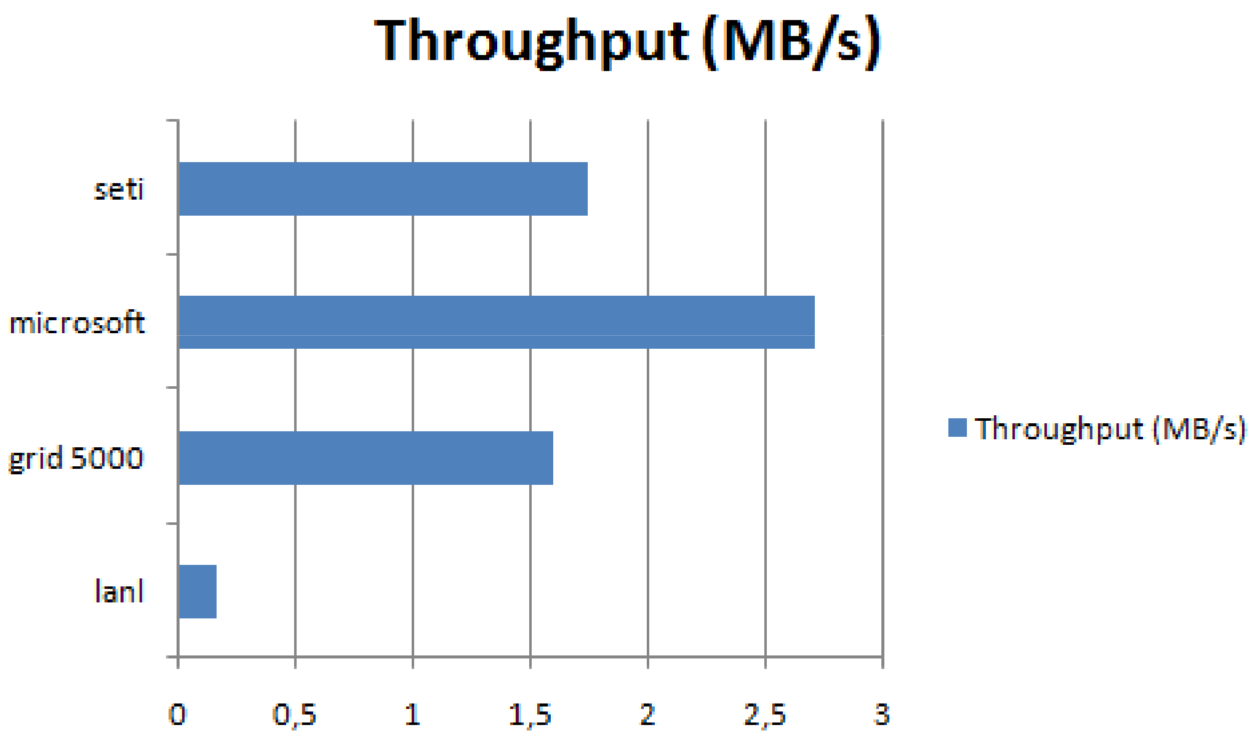
Our cluster

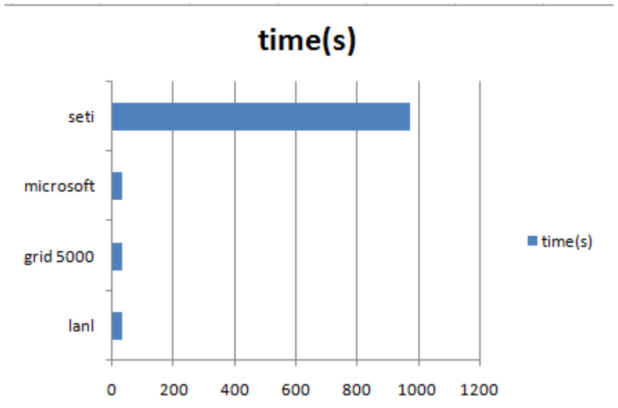

switch
gigabit eth link
namenode
job tracker
data node
task tracker
test scenario #1
test scenario #2

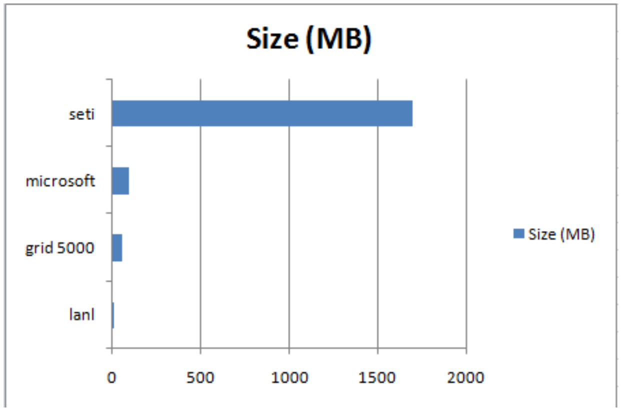
Tuning parameters

- MapReduce
  - Number of mappers
  - Number of reducers
  - Replication factor
- Pig – none
- MPI
  - Message length and frequence
  - Overlap IO and computation
  - Synchronization
  - MPI2: dynamic process creation, parallel IO

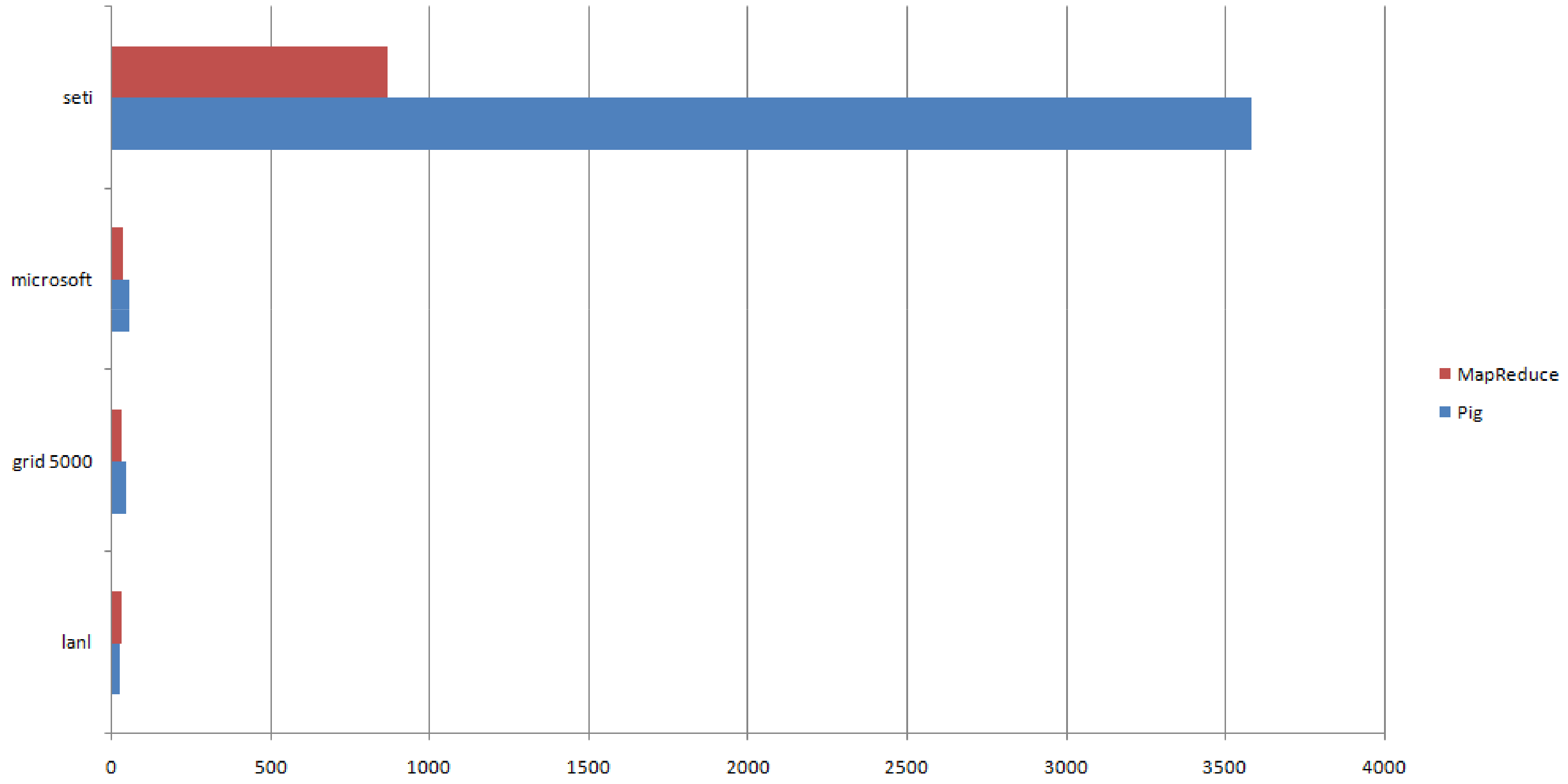
Results (1) – throughput



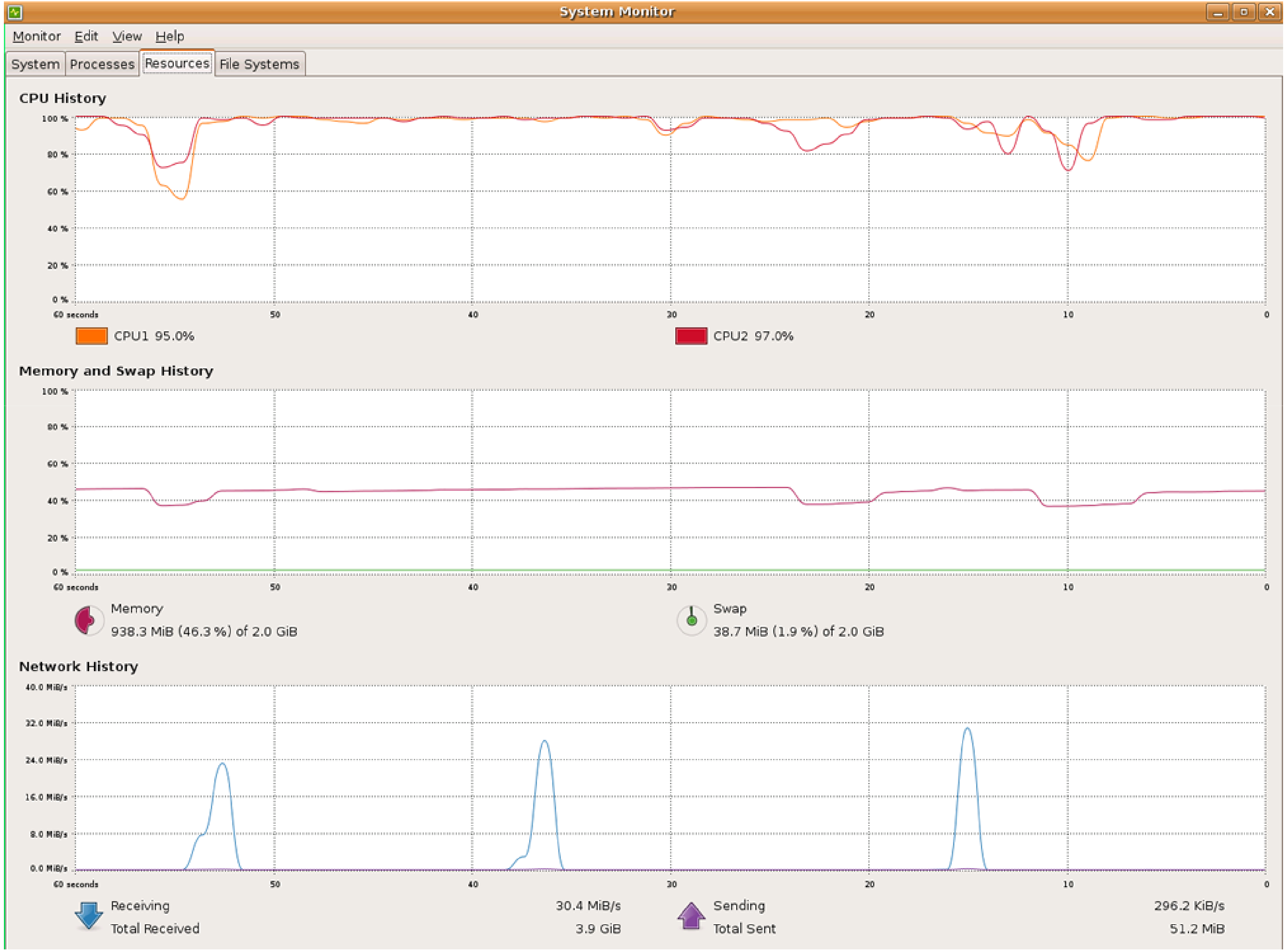




Results (2) – MapReduce vs Pig



Cluster utilization (1) – slaves at work



CPU History

🟧 CPU1 95.0%

🟥 CPU2 97.0%

Memory and Swap History

Memory
938.3 MiB (46.3 %) of 2.0 GiB

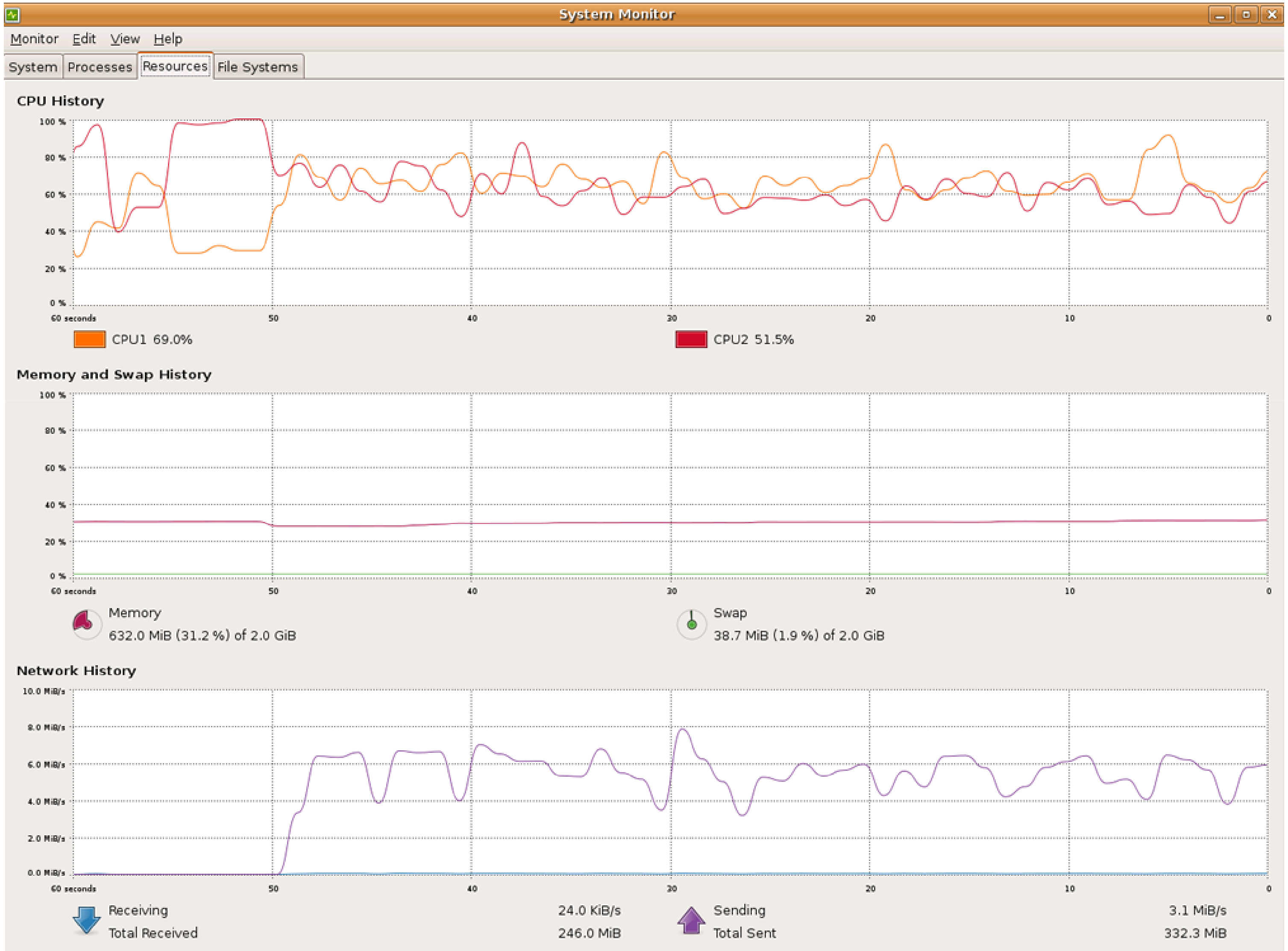
Swap
38.7 MiB (1.9 %) of 2.0 GiB

Network History

Receiving 30.4 MiB/s
Total Received 3.9 GiB

Sending 296.2 KiB/s
Total Sent 51.2 MiB

Cluster utilisation (2) – slaves at work



🟧 CPU1 69.0%

🟥 CPU2 51.5%

Memory
632.0 MiB (31.2 %) of 2.0 GiB

Swap
38.7 MiB (1.9 %) of 2.0 GiB

Receiving 24.0 KiB/s
Total Received 246.0 MiB

Sending 3.1 MiB/s
Total Sent 332.3 MiB

Conclusion

- Portability
  - Definitely Hadoop
  - MPI depends on RTS and implementation
- Productivity
  - MapReduce – an engineer's choice
  - Pig – fast development
  - MPI – too low-level, error-prone
- Scalability
  - Hadoop distributed fairly the jobs
  - MPI's communication might be a bottleneck

- Hadoop's HDFS is a big advantage over MPI

Questions ?

hadoop