In-situ MapReduce for Log Processing

Dionysios Logothetis, Kevin Webb, Kenneth Yocum
*UC San Diego*

Chris Trezzo
*Salesforce Inc.*

USENIX Annual Technical Conference
June 2011
Log analytics

• Data centers with 1000s of servers
• Generating logs with valuable information

• Data-intensive computing: Store and analyze TBs of logs

Examples:
• Click logs: ad-targeting, personalization
• Social media feeds: brand monitoring
• Purchase logs: fraud detection
• System logs: anomaly detection, debugging
Log analytics today

• “Store-first-query-later”
  – Migrate logs to dedicated clusters

Problems:
• Scale
  – e.g. Facebook collects 100TB a day!
  – Data migration stresses network and disks

• Failures
  – e.g. server is unreachable
  – Delay analysis or process incomplete data

• Timeliness
  – e.g. long data migration times
  – Hinders real-time apps: ad-targeting, fraud detection
In-situ MapReduce (iMR)

Idea:
• Move analysis to the servers
• MapReduce for continuous data
• Ability to trade fidelity for latency

Optimized for:
• Highly selective workloads
  – e.g. up to 80% data filtered or summarized!
• Online analytics
  – e.g. Ad re-targeting based on most recent clicks
An iMR query

The same:
• MapReduce API
  – map(r) → {k,v} : extract/filter data
  – reduce( {k, v[]} ) → v' : data aggregation
  – combine( {k, v[]} ) → v' : early, partial aggregation

The new:
• Provides continuous results
• Because logs are continuous
Continuous MapReduce

- iMR input is an infinite stream of logs

- Bound input with *sliding windows*:
  - Range of data
  - Update frequency
  - e.g. Process user clicks over the last 60’... 
    ... and update analysis every 15’

- Nodes output stream of results, one for each window

- Analysis continuously updated with new data
Processing windows in-network

- Aggregation trees for efficiency
  - Distribute processing load
  - Reduce network traffic

Problem:
- Overlapping data
  - Processed multiple times: wastes CPU
  - Sent to the root multiple times: wastes network
Efficient processing with *panes*

- Eliminate redundant work
- Divide window into *panes* (sub-windows)
- Each pane is processed and sent only once
- Root combines panes to produce window

- Saves CPU & network resources, faster analysis
Impact of data loss on analysis

- Servers may get overloaded or fail
- Apps may have latency requirements
- Data loss is unavoidable to ensure timeliness

Challenges:
- Characterize incomplete results
- Allow users to trade fidelity for latency
Quantifying data fidelity

• Data are naturally distributed across:
  – Space (server nodes)
  – Time (processing window)

• Panes describe temporal and spatial nature of data

• $C^2$ metric: annotates result windows with a “scoreboard”
  – Marks successfully received panes
Trading fidelity for latency

• Use $C^2$ spec to trade fidelity for latency

Users may specify:

• Maximum latency requirement
  – e.g. process window within 60sec

• Minimum fidelity
  – e.g. at least 50% of the total data

• Different ways to meet minimum fidelity
  – Impact latency and accuracy of analysis

• We identified 4 useful classes of $C^2$ specifications
Minimizing result latency

- Minimum fidelity with earlier results
  - e.g. 50% of the data
- Gives freedom to decrease latency
  - Returns the earliest data available
  - e.g. data from the fastest servers
- Appropriate for uniformly distributed events
  - Accurately summarizes relative event frequencies
Sampling non-uniform events

- Minimum fidelity with random sampling
  - e.g. random 50% of the data
- Less freedom to decrease latency
  - Included data may not be the first available
- Appropriate even for non-uniform data
  - Reproduces relative occurrence of events
Correlating events across time and space

• Leverage knowledge about data distribution

Temporal completeness:
• Include all data from a node or no data at all
  – e.g. all data from 50% of the nodes
• **Useful when events are local to a node**
  – e.g. counting events on a per node basis

Spatial completeness:
• Each pane contains data from all nodes
• **Useful for correlating events across servers**
  – e.g. click sessionization
Prototype

• Builds upon Mortar distributed stream processor
  [Logothetis et al., USENIX’08]
  – Sliding windows
  – In-network aggregation trees

• Extended to support:
  – MapReduce API
  – Paned-based processing
  – Fault tolerance mechanisms: operator restart, adaptive data routing
Processing data in-situ

- Analysis co-located with client-facing services
- Limited CPU resources for log analysis

- Goal: use available resources intelligently

- Load shedding mechanism
  - Nodes monitor local processing rate
  - Shed panes that cannot be processed on time

- Increases result fidelity under time and resource constraints
Evaluation

• System scalability

• Usefulness of $C^2$ metric
  – Understanding incomplete results
  – Trading fidelity for latency
  – Applications:
    • Click-stream sessionization
    • HDFS failure detection

• Processing data in-situ
  – Improving fidelity under load with load shedding
  – Minimize impact on services
Exploring fidelity-latency tradeoffs

• Hadoop DFS anomaly detection algorithm [Tan et al. WASL’08]

• Query: compute distribution of service times for every HDFS server, to detect outliers

• Data: HDFS log trace from 30-node cluster
Exploring fidelity-latency tradeoffs

- Data loss affects accuracy of distribution
  - Report: probability observed distribution is incorrect

- Temporal completeness
  - Distributions are 100% accurate
  - Computed on per server basis

- Spatial completeness & random sampling
  - Poor results if more than 20% data lost
  - Reduce latency by >25%

- $C^2$ allows to trade fidelity for lower latency
In-situ performance

- iMR side-by-side with a real service (Hadoop) on a 10-node cluster
- iMR executes a word count query
- Latency requirement set to 60sec.
- Vary CPU allocated to iMR (niceness)
- Report:
  - Result fidelity
  - Hadoop performance (job throughput)
- Shedding improves fidelity by 560%!
- Hadoop performance drops by <11%
- Little impact on Hadoop, while still delivering useful results
Conclusion

• In-situ architecture processes logs at the sources, avoids bulk data transfers, reduces analysis time

• Model allows incomplete data under failures or server load, provides timely analysis

• $C^2$ metric helps understand incomplete data and trade fidelity for latency

• Pro-actively sheds load, improves data fidelity under resource and time constraints