Scalable and Distributed Professor AT

Spyros Sioutas
PhD #6 Of AT
Associate Professor
Dept. of Informatics, Ionian University
ISD-Lab@DI [IU]
MMLab@CEID [Upatras]
CEID, UPATRAS, November ‘17
“Data is the new [crude] oil” (C. Humby, 2006)
“...and analytics is the combustion engine” (P. Sondergaard, 2011)
Elements of ‘Big Data’

• 1 – Data
• 2 – Compute Engines
• 3 – Distributed Datastores and Indexing
• 4 – Algorithms
• 5 – Privacy && Security
• 6 – Apps (business logic, workflows, policies)
Challenging issues

• **H**eterogeneity
  – In data sources, types, speed, accuracy, content,…
  – In data storage
  – In compute engines
  – In requirements

• **V**olatility
  – They all change fast!
What’s really Big?

• Data is big but...
  – Crunching them is getting faster and faster
  – More resources, bigger speeds, better algorithms

• Heterogeneity dramatically increases complexity in optimizing a workflow!
  – #runtimes, #datastores, #cost models, #policies
  – #datasets
How + where to execute my workflow?

Which technology to use? One to rule them all? If some new tech appears?

What is the ‘right’ data for me?

Type and size of resources to allocate? If it fails?

Optimize for different criteria

07/11/17, CEID, Upatras
Key observations

1. It’s not about big, but about the RIGHT data

Data-driven analytics

2. One size does NOT fit all

Multi-engine analytics
– Not size, but “relevance” counts
– How I could Discover the RIGHT data?
– By executing Large-Scale ML (Classification, Clustering, Prediction,...e.t.c.) Operators
– **Examples**: Content-based marketing, web advertising -- outliers indicate anomaly detection, fraud detection, Recommendation systems, Healthcare, Elasticity,..e.t.c.)
Prediction of Query Workload: We estimate the # VMs we have to Add/delete in time so as to continually satisfy SLAs → Elasticity

TIRAMOLA is the BEST EC Mechanism: It has been awarded in ACM SIGMOD and ACM/IEEE CCGRID.


2. Multi-engine analytics

- Set up the multi-engine cluster
- Build the network (p2p) overlay for indexing the right engines
- Type of Execution engines (MapReduce, StreamingProcessing,...)
- **Suitable Programming Language** for each engine
  - Call of MR functions in **Java** for Index Construction in **HBase engine**
  - Call of MR functions in **Scala / Python** for Index Construction in **Spark engine**
- Maintain the cluster by running periodically Load Balancing and Fault Tolerance operations.
Execution Engines and Complex Analytics Tasks

• Batch processing (MapReduce)
  – Disk based: Hadoop
  – In-memory: Spark
• Stream processing
  – Storm, S4, Flume, Kafka, etc.
• Iterative processing (Bulk Synchronous Parallel model)
  – Pregel, Hama, ...
• Interactive query processing
  – Dremel, Drill, Impala, etc.

Modern workflows can be long and complex:
  1. Multiple data types from different sources
  2. Diverse operators (ML, NLP, image processing, bio, user-defined, etc.)
  3. Under various constraints/policies
Data Stores and **Data Structures**

- Relational datastores
- NoSQL Datastores
  - Key-value, Document, wide-column, graph, ...
- Column-stores (f.e. BIG GOOGLE TABLES)
- Main memory, disk based, hybrid, ...
- **P2P or Decentralized Data Structures:**
  - **A** DHT-based (**Hash-based**)
  - **B** Hierarchical-Based (**tree-based**)
  - **C** Skip-List-Based (**Probabilistic**)

07/11/17, CEID, Upatras
Big Data Technology is based on Hadoop Distributed File System (HDFS)

- Files split into 128MB blocks
- Blocks replicated across several datanodes (often 3)
- Namenode stores metadata (file names, locations, etc)
- Files are append-only

MapReduce Parallel Programming framework:
- 1. Runs jobs submitted by users.
- 3. Colocated with file system.
Typical Hadoop Cluster

- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth in rack, 8 Gbps out of rack
- Node specs (Facebook): 8-16 cores, 32-48 GB RAM, 10×2TB disks

**Challenges:**
- System churn: machines can fail or exit the system any time
- Scalability: need to scale to 10s or 100s of thousands machines
- Heterogeneity:
  - Latency: 1ms to 1000ms
  - Bandwidth: 32Kb/s to 100Mb/s
  - Nodes stay in system from 10s to a year
[A]. Distributed Hash Tables (DHTs)

- Distribute (partition) a hash table data structure across a large number of servers
  - Also called, \textit{key-value store}

- \textbf{Key identifier = SHA-1(key), Node identifier = SHA-1(IP address)}
- \textbf{Each key\_id is mapped to the node\_id with the smallest node\_id $\geq$ key\_id}

- Two operations
  - \texttt{put(key, data); // insert “data” identified by “key”: Where? Lookup!!}
  - \texttt{data = get(key); // get data associated to “key”: From where? Lookup!!}
[A]. The lookup problem

Tell me the route from N6 to N2
The shortest, the better!!!
[A]. Chord Lookup Service (Protocol)

- Consistent Hash Function Method assigns each node and each key an m-bit identifier using SHA 1 (Secure Hash Standard).

  \[ m = \text{any number big enough to make collisions improbable} \]
  
  Key identifier = SHA-1(key)
  Node identifier = SHA-1(IP address)

- Both are uniformly distributed

- Both exist in the same ID space \(0..2^{m-1}\)
  - This space has been partitioned across N machines
  - Each key is mapped to the node with the smallest largest id (consistent hashing)
[A]. The Chord algorithm – Scalable node localization

- Identifiers are arranged on a identifier circle modulo $2^m$ => Chord ring
- The key $k$ is assigned to the node whose identifier is equal to or greater than the key’s identifier
- This node is called successor($k$) and is the first node clockwise from $k$.

Finger table:
\[ \text{finger}[i] = \text{successor} (n + 2^{i-1}) \]

Each node stores information about only a small number of nodes ($m$)

$\Rightarrow$ Number of messages $O(\log N)!$
[A]. The Chord algorithm – Node joins and stabilization

- To ensure correct lookups, all successor pointers must be up to date.
- stabilization protocol running periodically in the background and Updates finger tables and successor pointers.
- No influence on performance as long as fingers are adjusted faster than the network doubles in size.
Massive Failure of nodes

- Correctness relies on correct successor pointers
- What happens, if N14, N21, N32 fail simultaneously?
- How can N8 acquire N38 as successor?

- Each node maintains a successor list of size r.
- If the network is initially stable, and every node fails with probability $\frac{1}{2}$, the expected time to find successor is $O(\log N)$.

Massive failures have little impact

Failed Lookups (Percent)

Failed Nodes (Percent)
The Example of “Cassandra” Distributed NoSQL DB (Facebook)

• N1, N2…..Nx are computing nodes of the same rack
• M1, M2…..My are computing nodes of the same rack
• Each rack is structured as a Chord overlay-network
• The whole CLUSTER (CLOUD) is structured as a Chord overlay Between rack-switches (Each rack-switch talks directly to it’s Master node)
[B]. BATON Architecture

- Given a node $x$, we say that the node immediately prior to it in the IN-ORDER (LEFT-ROOT-RIGHT) traversal is *left adjacent* to it, and the node immediately after $x$ is *right adjacent* to it.
- *Adjacent nodes may be at very different levels*
- Routing Tables determine the node-neighborhoods.
- *A tree is balanced if and only if at any node in the tree, the height any of its two subtrees differ by at most one.*

Binary Balanced Tree Index Architecture
[B]. Index construction

- Each node is assigned a range of values

- The range is:
  - Greater than the range managed by its left adjacent node
  - Smaller than the range managed by its right adjacent node
[B]. Exact-Match and Range Queries in $O(\log N)$ and $O(\log N+K)$

- Example: node $h$ wants to search/insert/delete data belonged to node $c$, say 74.
- Follow neighbor-links / adjacent links
- $K$ is the total number of nodes containing searched results
Load balancing--Node Join&&Node Departure--Fault Tolerance

- **Load balancing** process is initialized when a node is overloaded or under loaded due to insertion or deletion. We transfer sub-loads to neighbor/adjacent nodes $\rightarrow$ $O(\log N)$ hops

- **Node Join/leave operations** require $O(\log N)$ hops since exploit the logarithmic length of neighbor-links and adjacent –links.

- **Fault Tolerance after node’s failure** (node’s departure) $\rightarrow$ Restructuring + Load Balancing: It costs much more than logarithmic...
[B]. BATON*

• Why does BATON limit to a binary tree structure? Why not a multi-way tree structure?
  – Multi-way tree structure breaks “in order” traversal.
  – Not straightforward to expand the structure of routing tables.
[B]. A solution in BATON*

• Each node (peer) manages not only one range of values but \( m-1 \) ranges of values and \( m \) children as in the traditional multi-way tree.

• Range of values is:
  - less than ranges managed by the first \( m/2 \) children nodes
  - greater than ranges managed by the last \( m/2 \) children nodes.

• Neighbor nodes are nodes in distance
  - \( d \times m^i \) (\( d=1..m-1 \))
**Theorem**: The maximum size of a routing table of a node at level \( l \) is \( m \times l \)

**Corollary**: The maximum size of a routing table of a node in the network is \( m \times \log_m N \)
[B]. Benefit vs cost

• Benefit:
  – Search, insertion, deletion:
    • BATON*: $O(\log_m N)$
    • BATON: $O(\log_2 N)$
  – Fault tolerance: BATON* is better because of keeping more links.
  – Load balancing: BATON* is better because of having more leaf nodes. The later implies that is much easier to do load balancing.

• Cost:
  – Cost of updating routing tables:
    • BATON*: $O((m – 1) \cdot \log_m N)$
    • BATON: $O(\log_2 N)$
[C.] Skip List – based P2P structures

- Each box with an arrow represents a pointer and a row is a linked list giving a sparse subsequence.

- The numbered boxes (in yellow) at the bottom represent the ordered data sequence.

- Searching proceeds downwards from the sparsest subsequence at the top until consecutive elements bracketing the search element are found.
[C.] Skip Graphs

- A major distinction from DHTs (Chord e.t.c..) is that there is no hashing of search keys of resources, which allows related resources to be near each other.

- This property makes searches for values within a given range feasible.

- Another strength of skip graphs is the resilience to node failure in both random and adversarial failure models.
EXAMPLE of HOW TO USE Distributed Data Structures in SQL-based workflows

- SQL is a top-ranked data scientist skill
- Tables frequently stored in different locations
- Need for concurrent access to big data

07/11/17, CEID, Upatras
Query Processing – Example
(Execution Plan1: Local && P2P Indexes )

SELECT *
FROM customer, nation, orders, lineitem, part, partsupp
WHERE c_nationkey = n_nationkey
AND c_orderkey = o_orderkey
AND o_orderkey = l_orderkey
AND l_partkey = p_partkey
AND p_partkey = ps_partkey
AND p_retailprice > 2090
AND n_name = 'GERMANY'

These three external joins require a P2P (network overlay) index like CHORD, BATON*, SKIP-GRAPH, ..e.t.c.
Execution Plan 2:
Local Indexes && High Streaming Velocity
Move the sub-results that produced in parallel
to a Spark Engine

SQL

PostgreSQL

memsql

Spark SQL

customer
nation

part
partsupp

lineitem
orders
Query Graph → Multi – Engine Planner

**Spark SQL**

```
SELECT c_name, o_orderdate
FROM inter1, inter2, orders, lineitem
WHERE l_partkey = p_partkey
And o_custkey = c_custkey
And o_orderkey = l_orderkey
```

- **Exec Time**: 33.2 sec
- **Rows**: 824

**MemSQL: inter1**

```
SELECT p_partkey
FROM part, partsupp
WHERE p_partkey = ps_partkey
and p_retailprice > 2090
```

- **Exec Time**: 14.58 sec
- **Rows**: 900

**Postgres: inter2**

```
SELECT c_custkey, c_name
FROM customer, nation
WHERE c_nationkey = n_nationkey
and n_name = 'GERMANY'
```

- **Exec Time**: 82 ms
- **Rows**: 30182

07/11/17, CEID, Upatras
Our team has developed new scalable distributed data structures [ART, ART+, D²-tree and D³-tree] that outperform all the previous fundamental structures.

- **BATON, BATON*, P-Ring, D²-Tree, D³-Tree**

<table>
<thead>
<tr>
<th>Structures</th>
<th>Search</th>
<th>Search with massive failures</th>
<th>Node Updates (updating rout. tables)</th>
<th>Element Updates (load balancing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATON</td>
<td>$O(\log N)$</td>
<td>— Yes</td>
<td>$\overline{O}(\log N)$</td>
<td>$\overline{O}(\log N)$</td>
</tr>
<tr>
<td>BATON*</td>
<td>$O(\log_m N)$</td>
<td>— Yes</td>
<td>$\overline{O}(m \cdot \log_m N)$</td>
<td>$\overline{O}(m \cdot \log_m N)$</td>
</tr>
<tr>
<td>P-Ring</td>
<td>$O(\log_d N)$</td>
<td>— Yes</td>
<td>$\widetilde{O}(d \cdot \log_d N)$</td>
<td>$\widetilde{O}(d \cdot \log_d N)$</td>
</tr>
<tr>
<td>D²-Tree</td>
<td>$O(\log N)$</td>
<td>— No</td>
<td>$\widetilde{O}(\log N)$</td>
<td>$\widetilde{O}(\log N)$</td>
</tr>
<tr>
<td>D³-Tree</td>
<td>$O(\log N)$</td>
<td>$\widetilde{O}(\log N)$ Yes</td>
<td>$\widetilde{O}(\log N)$</td>
<td>$\widetilde{O}(\log N)$</td>
</tr>
</tbody>
</table>

$N$: number of nodes; $m$: fanout; $d$: order of the ring; $\widetilde{O}$: amortized bound; $\overline{O}$: expected amortized bound; Theor: theoretical bound; Exp: empirical evidence.
#### Brief Comparison

- **BATON, BATON**, P-Ring, D\(^3\)-Tree, ART, ART+

<table>
<thead>
<tr>
<th>Structures</th>
<th>Search Key</th>
<th>Insert/Delete key (load-balancing)</th>
<th>Max. Size of Routing Table</th>
<th>Join/Depart Peer (updating routing tables)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATON</td>
<td>(O(\log N))</td>
<td>(\tilde{O}(\log N))</td>
<td>(O(\log N))</td>
<td>(\tilde{O}(\log N))</td>
</tr>
<tr>
<td>BATON(^*)</td>
<td>(O(\log_m N))</td>
<td>(\tilde{O}(m \cdot \log_m N))</td>
<td>(O(m \cdot \log_m N))</td>
<td>(\tilde{O}(m \cdot \log_m N))</td>
</tr>
<tr>
<td>P-Ring</td>
<td>(O(\log_d N))</td>
<td>(\tilde{O}(d \cdot \log_d N))</td>
<td>(O(\log N))</td>
<td>(\tilde{O}(d \cdot \log_d N))</td>
</tr>
<tr>
<td>D(^3)-Tree</td>
<td>(O(\log N))</td>
<td>(\tilde{O}(\log N))</td>
<td>(O(\log N))</td>
<td>(\tilde{O}(\log N))</td>
</tr>
<tr>
<td>ART</td>
<td>(\hat{O}(\log^2 b \log N))</td>
<td>(\tilde{O}(m \cdot \log_m \log N))</td>
<td>(O(N^{1/4} / \log^c N))</td>
<td>(\tilde{O}(m \cdot \log_m \log N))</td>
</tr>
<tr>
<td>ART(^+)</td>
<td>(\hat{O}(\log^2 b \log N))</td>
<td>(\tilde{O}(\log \log N))</td>
<td>(O(N^{1/4} / \log^c N))</td>
<td>(\tilde{O}(\log \log N))</td>
</tr>
</tbody>
</table>

\(N\): number of peers, \(m\): fanout, \(d\): order of the ring, \(c > 0\), \(b\): double-exponentially power of 2,
\(\hat{O}\): expected bound, \(\tilde{O}\): amortized bound, \(\widetilde{O}\): expected amortized bound
Brief Comparison

<table>
<thead>
<tr>
<th>P2P architectures</th>
<th>Lookup key</th>
<th>Insert/delete key</th>
<th>Maximum Size of routing table</th>
<th>Join/Depart node</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHORD</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$ w.h.p.</td>
</tr>
<tr>
<td>H-F-Chord(a)</td>
<td>$O(\log N \log \log N)$</td>
<td>$O(\log N \log \log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>LPRS-Chord</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Skip Graphs</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(1)$</td>
<td>$O(\log N)$ amortized</td>
</tr>
<tr>
<td>BATON</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$ w.h.p.</td>
</tr>
<tr>
<td>BATON*</td>
<td>$O(\log_m N)$</td>
<td>$O(\log_m N)$</td>
<td>$O(m \log_m N)$</td>
<td>$O(m \log_m N)$</td>
</tr>
<tr>
<td>ART-tree</td>
<td>$O(\log^2 N \log N)$</td>
<td>$O(\log^2 N \log N)$</td>
<td>$O(N^{1/4} / \log^c N)$</td>
<td>$O(\log \log N)$ expected w.h.p.</td>
</tr>
</tbody>
</table>

Table 1: Performance comparison between ART, Chord, BATON and Skip Graphs.


Thank you!

“THAT’S your Ark for the Big Data flood? Noah, you will need a lot more storage space!”

I could also urge him: “Noah, please, think decentralized”
http://www.ionio.gr/~sioutas

07/11/17, CEID, Upatras