Do we need Rack-Scale Coordination?

Alysson Bessani
Rack-Scale Computers (RSC)
(or Datacenter-in-a-Box systems)

• Tightly integrated rack (in a single box)
• Very fast node interconnection
• Special-purpose components
• “Uncommon” network topologies
Rack-Scale Computers (RSC) (or *Datacenter-in-a-Box* systems)

“Traditional” Model

```
Node ── Node ── Node
```

“Torus” Model

```
Node ── Node ── Node
      └── Node ── Node ── Node
```
Do they need coordination?

- Leader election
- Locks
- Barriers
- Atomic counters
- Augmented Queues
  ...
- Configuration management
Out of the box Alternatives

• Shared memory algorithms

• Multi-kernel coordination

• Datacenter coordination
Single-machine Coordination

• **Shared-memory algorithms**
  – Classical shared memory locking algorithms exist since the 70s (Lamport’s Bakery, etc.)
  – Algorithms require some consistency on the shared memory
    • Total Store Ordering (TSO – weaker than sequential consistency)
    • The best know result requires a constant number of remote memory references and memory barriers [PODC’13]

• **Multi-kernel Solution**
  – A service (deployed on a core) that provides all the coordination primitives that applications need
    • E.g., Barreelfish supports a service like Zookeeper [APSys’12]

• **Both solutions do not tolerate faults**
Datacenter Coordination

- Coordination services:

<table>
<thead>
<tr>
<th>System</th>
<th>Data Model</th>
<th>Sync. Primitive</th>
<th>Wait-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boxwood [44]</td>
<td>Key-Value store</td>
<td>Locks</td>
<td>No</td>
</tr>
<tr>
<td>Chubby [17]</td>
<td>(Small) File system</td>
<td>Locks</td>
<td>No</td>
</tr>
<tr>
<td>Sinfonia [6]</td>
<td>Key-Value store</td>
<td>Microtransactions</td>
<td>Yes</td>
</tr>
<tr>
<td>DepSpace [14]</td>
<td>Tuple space</td>
<td>cas/replace ops</td>
<td>Yes</td>
</tr>
<tr>
<td>ZooKeeper [31]</td>
<td>Hierar. of data nodes</td>
<td>Sequencers</td>
<td>Yes</td>
</tr>
<tr>
<td>etcd [3]</td>
<td>Hierar. of data nodes</td>
<td>Sequen./Atomic ops</td>
<td>Yes</td>
</tr>
<tr>
<td>LogCabin [5]</td>
<td>Hierar. of data nodes</td>
<td>Conditions</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- dependable (limited) storage
- synchronization power
- client failure detection
So…

• A RSC has multiple fault domains, so fault tolerance is needed
  – Coordination services are our best bet

• **Durability** may or may not be needed
  – Strictly required for configuration management

• **Extensibility** for improved performance
  – See the “Extensible Distributed Coordination” paper/talk on EuroSys’15
Traditional Network

• The coordination service is implemented as usual, i.e., “just deploy Zookeeper on your RSC”
  – A bunch of replicas ensure the service is fault tolerant
  – Durability techniques ensure full crash recovery

• Possible improvements:
  – More efficient replication algorithms
    • DARE [HPDC’15] proposes RAFT-like RDMA-based state machine replication with 12 microsec latency (1kB write)
      – 35x faster than ZK in the same network
  – Faster durability mechanisms (e.g., NVRAM)
Torus Network

• Coordination scope
  – L0: local CPU
  – L1: CPU + other local computing devices
  – L2: all nodes reachable in one hop
  – L3: all nodes reachable in two hops
  – …
  – LN: all nodes reachable in N-1 hops
• This may lead to the development of new quorum systems and fault-tolerant algorithms
Questions… questions…

• The RSC software stack requires general coordination support. The question is:
  – Do we need anything specific or it is just a matter of deploying what we already have?

• Other questions:
  – Can specialized hardware (FPGA) help?
  – Can we assume/implement reliable failure detection?
  – Efficiency or predictability?
  – What about data-centric coordination?
More Questions?