Akka Distributed Data

Ryan Knight
Principal Architect
@knight_cloud

Agenda

- Challenges of Distributed Systems
- Conflict Free Replicated Data Types - CRDTs
- Akka Clustering
- Akka Distributed Data

Challenges of Distributed Systems

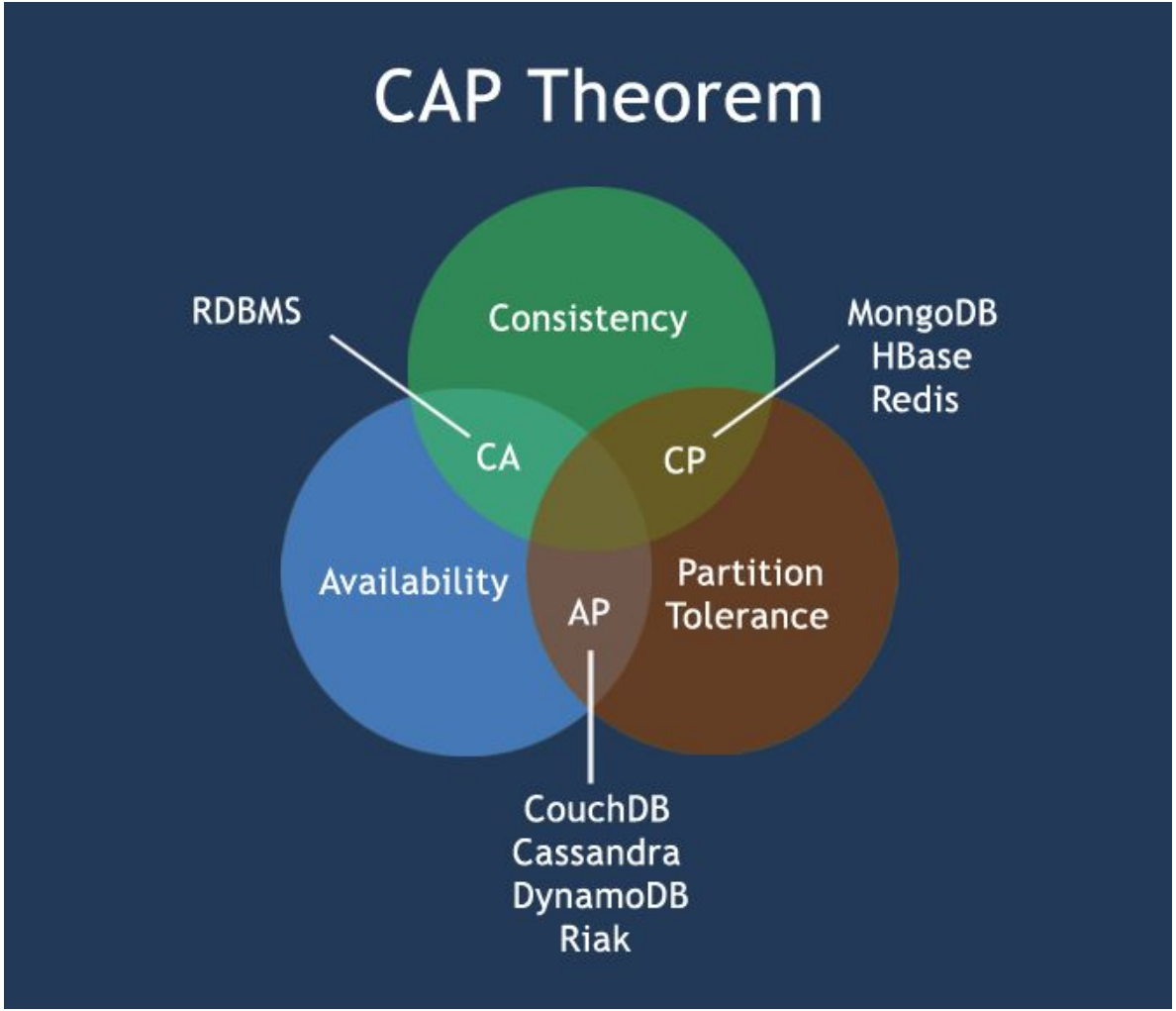
Challenges of Distributed Computing

- Replication is Slow
- Servers Fail
- The network is not reliable
- Latency > 0
- Limited Bandwidth

Fallacies of Traditional Data Models

- Total Global Ordering is not possible
- Data is not a single opaque value
- ACID Transactions are not possible

Global Locks / Distributed Transactions


CAP Theorem
RDBMS
Consistency
MongoDB
HBase
Redis
CA
CP
Availability
Partition
Tolerance
AP
CouchDB
Cassandra
DynamoDB
Riak

Eventual Consistency (EC)

- Embracing failure in distributed systems
- Reconciling different operation orders
- EC with Probabilistic Guarantees
- EC with Strong Guarantees

Conflict Free Replicated Data Types - CRDTs

Avoiding Conflicts

What are CRDTs

Data types that guarantee convergence to the same value in spite of network delays, partitions and message reordering

http://book.mixu.net/distsys/eventual.html

Rethinking how we view Data

- Not just a place to dump values
- Abstraction of the data type
- Data Structure that tells how to build the value

Why CRDTs

- Replicate data across the network without any synchronization mechanism
- Avoid distributed locks, two-phase commit, etc.
- Consistency without Consensus

Value of CRDTs

- Sacrifice linearizability (guaranteed ordering) while remaining correct
- Used to build AP Architectures - Highly Available and Partition Tolerant

Monotonic Sequence

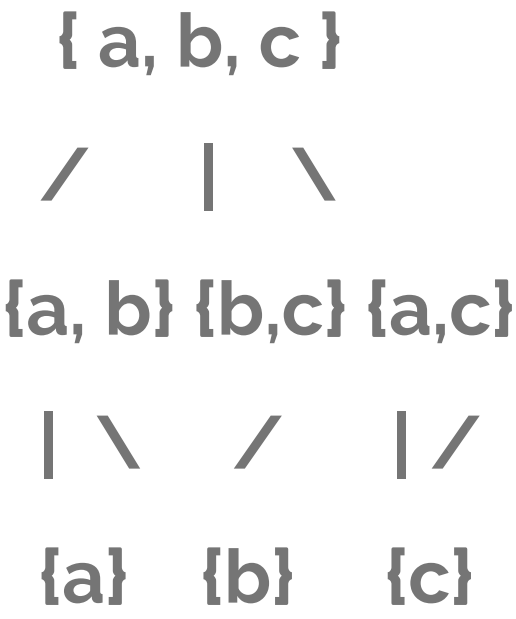
- Monotonic Sequences - Sequence that always increases or always decreases
- Monotonic Sequences are eventually consistent without any need for coordination protocols

Convergent Operations

- Associative (a+(b+c)=(a+b)+c) - grouping doesn't matter
- Commutative (a+b=b+a) - order of application doesn't matter
- Idempotent (a+a=a) - duplication does not matter

Example Operations

Union (Items)



Max Values

CRDT Counters

- Grow-only counter - only supports increments
- Positive-negative counter
  - Two grow counters, one for increments and another for decrements

CRDT Registers

- Last Write Wins Register
  - Cassandra Columns
- Multi-valued -register
  - Objects (values) in Riak

CRDT Sets

- Grow-only set -> merge by union(items) with no removal
- ORSet (Observer /Remove) - uses version vector and birth dots.
  - Once removed, an element cannot be re-added
  - Version vector and the dots are used by the merge function

CRDT Maps

- ORMap
- ORMultiMap
- LWWMap
- PNCounterMap

CRDT Compose

- CRDT Value can contain another CRDT
- ORSet can contain a G-Counter
- ORMap can contain a LWW Register

CRDT Implementations

- Riak Data Types are convergent replicated data types
  - https://docs.basho.com/riak/kv/2.2.0/learn/concepts/crdts/
- SoundCloud Roshi
  - https://github.com/soundcloud/roshi
- Akka Distributed Data

Akka Clustering

Akka

- Actor Based Toolkit
- Simple Concurrency & Distribution
- Error Handling and Self-Healing
- Elastic and Decentralized
- Adaptive Load Balancing

What is an Actor

- Isolated lightweight processes
- Message Based / Event Driven
- Run Asynchronously
- Processes one message at a time
- Sane Concurrency
- Isolated Failure Handling

Actor Systems

- Actor system is the hierarchy of collaborating actors
- Parent actors delegate work to child actors
- Child actors are supervised by Parent Actors
- Failure can be propagated back up Actor Hierarchy

Akka Clustering

- Peer-to-peer based cluster membership
- Communicates state via gossip protocols
- No single point of failure or single point of bottleneck.
- Automatic node failure detector

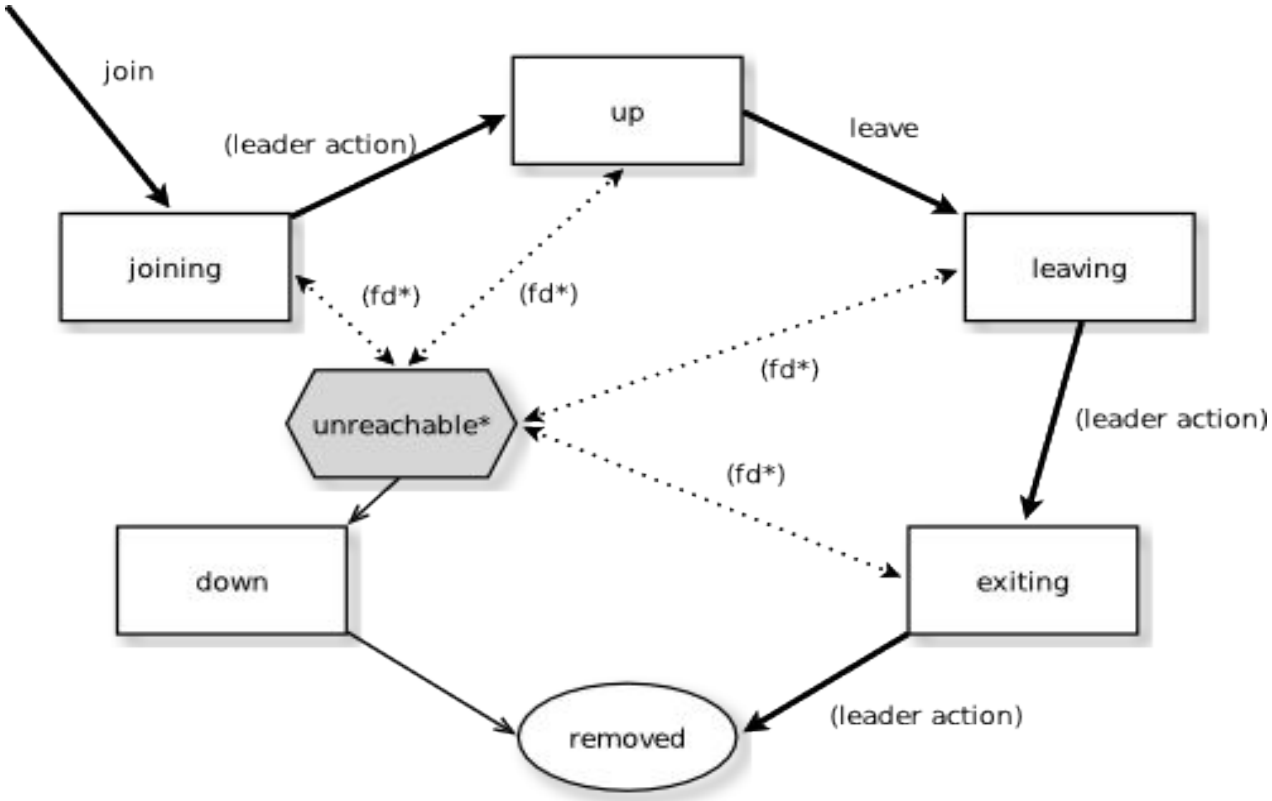
Gossip Protocol

- Sharing state by gossiping with neighbors
- Each node holds state and picks a random node to share information with
- Reliable communication is not assumed

Akka Clustering

- Cluster Singletons
- Cluster Roles
- Cluster Events
- Cluster-Aware Routers
- Cluster Sharding

Akka Cluster State - Monotonic!



Akka Distributed Data

Akka Distributed Data

- Replicated in-memory data store
- Share Data between Akka Cluster Nodes
- Low latency and high-availability
- Key-Value store like API

State Based CRDTs

- Akka Distributed Data only supports state based CvRDT's
- Require storage of extra data to facilitate merging
- Entire State of CRDT's must be disseminated

Delta State Based CRDTs

- Akka 2.5 Introduced Delta State CRDTS
- Only recently applied mutations to a state are disseminated instead of the entire state

Data Resolution

- Concurrent updates automatically resolved with monotonic merge function
- Fine Grained Control of Consistency Level of Reads and Writes
- Update from any node without coordination

Fine Grained Control of Consistency

- WriteLocal, WriteTo(n), WriteMajority, WriteAll
- ReadLocal, ReadFrom, ReadMajority, ReadAll
- Majority is N/2 + 1
- Guaranteed Consistency
  - (nodes_written + nodes_read) > N

Data Distribution

- Data Spread two ways depending on Consistency Level
- Direct replication to meet Consistency Level of Write
- Gossip dissemination to remaining nodes

Data Types

- Implements the ReplicatedData Trait
  - Monotonic merge function
- Counters: GCounter, PNCounter
- Sets: GSet, ORSet
- Maps: ORMap, ORMultiMap, LWWMap, PNCounterMap
- Registers: LWWRegister, Flag

Replicated Data Type Scala Interface

```
trait ReplicatedData {
  type T <: ReplicatedData
  /**
   * Monotonic merge function.
   */
  def merge(that: T): T
}
```

Replicated Data Type Java Interface

```
public class TwoPhaseSet extends AbstractReplicatedData<TwoPhaseSet> {
  public final GSet<String> adds;
  public final GSet<String> removals;

  public TwoPhaseSet mergeData(TwoPhaseSet that) {
    return new TwoPhaseSet(this.adds.merge(that.adds),
        this.removals.merge(that.removals));
  }
}
```

The Replicator Actor

- Performs all Replication
- Started on all cluster nodes participating in Distributed Data
- The replicator is similar to a key-value store:
  - Keys are strings, values are ReplicatedData
- Data is replicated directly and via gossiping

The Local Replicator

```
val replicator = DistributedData(context.system).replicator
```

- All Communication is done via the local replicator
- Accessed via the DataReplication extension
- Supported operations are Get, Subscribe, Update and Delete

Updating

- Key typed with the distributed data type
- Initial value
- Write consistency -> Once met sends an UpdateSuccess message back
- Optional request context - used to send response to the sender on UpdateSuccess message
- Update function

Update Example

```
Update<LWWMap<LineItem>> update = new Update<>(dataKey,
LWWMap.create(), writeMajority,
        cart -> updateCart(cart, add.item));

replicator.tell(update, self());
```

Change Notifications

- Subscription is done by sending a Subscribe message to the local replicator
- The actor will then receive changed messages

Change Notifications

```
case c @ Changed(DataKey) =>
      val data = c.get(DataKey)
      println()
      println("Current elements:")
      data.entries.foreach(println)
```

Pruning Algorithm

- When a node is removed from the cluster a pruning algorithm is used to collapse data

Additional Resources

- http://doc.akka.io/docs/akka/snapshot/scala/distributed-data.html
- Strong Eventual Consistency and Conflict-free Replicated Data Types talk by Mark Shapiro
  - http://research.microsoft.com/apps/video/default.aspx?id=153540&r=1
- http://book.mixu.net/distsys/eventual.html
- https://www.infoq.com/presentations/crdt-soundcloud?utm_source=infoq&utm_medium=slideshare&utm_campaign=slidesharesf

Questions?