Infrastructure Technologies for Large-Scale Service-Oriented Systems

Kostas Magoutis
magoutis@csd.uoc.gr
http://www.csd.uoc.gr/~magoutis
Amazon e-commerce platform

- World-wide operation, several data centers
- Tens of millions of customers at peak times
- Strict operational requirements
  - Performance, reliability, efficiency, continuous growth
- Reliability, availability amongst most important reqs
  - Dependent upon how application state is managed
- Need “always-writable” store, despite failures
Assumptions and requirements

• Query model
  – Simple put/get interface
  – Primary key, single-entry access only

• ACID properties
  – Strict ACID not needed
  – Must trade consistency for availability

• Efficiency
  – Must support stringent SLAs
  – Configurability is important to achieve SLAs

• Other
  – Secure environment
  – High scalability goals
Amazon service-oriented architecture

- Rendering service may construct its response by sending requests to over 150 other services
- Stateless but may use caching
- Each service in the call chain must obey performance contract
Service-level agreements

• Specification
  – System agrees to maintain level of service (e.g., response time) if client does not exceed load threshold
  – E.g., <300ms for 99.9% of requests if load <500 reqs/sec

• Amazon differs from rest of the industry
  – SLA targeted towards all customers, not just majority
  – Using 99.9% rather than average or median

• To enforce SLA, system needs feedback loop
  – Monitoring
  – Admission control
  – Resource reservation
Key partitioning and replication

Nodes B, C, and D store keys in range (A, B) including K.
Consistency

- **Eventual consistency**
  - When to resolve and who resolves conflicts
  - How to resolve conflict

- **Quorum**
  - Parameterized
  - How does it differ from traditional quorum

- **Hinted handoff**
  - Maintain replication level despite failures

- **Synchronization protocol ("anti-entropy")**
Versioning via vector clocks

\[ D1 ([Sx,1]) \]

\[ \text{write} \]
\[ \text{handled by } Sx \]

\[ \downarrow \]

\[ D2 ([Sx,2]) \]

\[ \text{write} \]
\[ \text{handled by } Sx \]

\[ \downarrow \]

\[ D3 ([Sx,2],[Sy,1]) \]

\[ \text{write} \]
\[ \text{handled by } Sy \]

\[ \downarrow \]

\[ D4 ([Sx,2],[Sz,1]) \]

\[ \text{write} \]
\[ \text{handled by } Sz \]

\[ \downarrow \]

\[ D5 ([Sx,3],[Sy,1],[Sz,1]) \]

\[ \text{reconciled and written by } Sx \]
Synchronization protocol: Merkle trees
Membership and failure

- Membership information distributed via gossip
  - This includes partitioning and placement information
  - Each node forwards key’s read/write operations directly

- Transient failure detected by inability to communicate
  - No repartitioning during transient failure

- Explicit node join and leave operations
  - Using a seed node

- No need for decentralized failure detection protocols
Implementation

• Each storage node has three main components
  – Request coordination (a state machine for each operation)
  – Membership and failure detection
  – Local persistence engine (BDB, MySQL, mem+disk)

• Request coordination
  – Send request to nodes in preference list
  – Wait for minimum number of required responses
  – If too few replies within time bound, fail request
  – Gather all versions, determine which to return
  – Perform syntactic reconciliation, update vector clock

• Read repair
  – Update stale replicas, in-band synchronization
Balancing performance and durability

(hourly plot of latencies during our peak season in Dec. 2006)
Other lessons learned

• Divergent versions: when and how many
  – Increase mostly due to concurrent writes

• Client-driven vs. server-driven coordination
  – Avoid forwarding step, need for load balancer

• Balancing foreground vs. background tasks
  – Admission control and monitoring to achieve SLAs