Enforcing Routing Consistency in Structured Peer-to-Peer Overlays: Should We and Could We?

Wei Chen and Xuezheng Liu
Microsoft Research Asia
Routing Consistency

- Key-based routing (KBR)
  - Large key space
  - Routing to a destination close to a given key
- KBR consistency: routings always reach the owner of the key.
Why Routing Consistency?

- Important to applications (p2p storage, pub/sub, etc). Without consistency:
  - Incorrect/failed results
  - Performance cost: extra retries, extra maintenance
  - Application complexity
    ⇒ limit applicability of structured p2p overlay
- Prior researches focus on performance and scalability
How to Enforce Routing Consistency?

- Formal specification
- Algorithm for strong consistency
  - Provably correct
  - Use group membership service (GMS)

Consistency Spectrum:

- Weakest: Best-effort P2P protocols (Pastry, Tapestry)
- Strongest: Group Membership Service (GMS)

- Weakly consistent KBR
- Strongly consistent KBR
Rest of the Talk

- Introduction
- System model
- Group membership service
- Routing consistency specification
- Algorithm for strong consistency
- Proposals for scalability and adaptivity
- Ongoing and future work
System Model

- Nodes $\Sigma = \{x_1, x_2, x_3, \ldots\}$
- Time $\mathcal{T} = \{0, 1, 2, \ldots\}$
- Nodes may join, leave/crash.
- Membership pattern $\Pi = \mathcal{T} \rightarrow 2^\Sigma$
  - $\Pi(t)$: finite, set of live nodes at time $t$.
  - $sset(\Pi)$: the set of eventual live nodes, if $\Pi$ is eventually stable.
- Asynchronous message passing
  - No creation, no duplication, reliable.
Group Membership Service (GMS)

- Membership view $v = (set, ver)$
  - $v.set \subseteq \Sigma$, $v.ver$ is a nonnegative integer
- Query interface action: `getCV()`
  - Return a view $v$ or a $\perp$
- Internal update actions: `join()` and `remove()`
- `getCV()`, `join()` and `remove()` are totally ordered
GMS Properties

- those related to KBR

- **Causality Consistency**: If node $x_1$ and $x_2$ each invokes a getCV(), and the return of getCV() on $x_1$ causally precedes the invocation of getCV() on $x_2$, and the return values are two views $v_1$ and $v_2$, respectively, then $v_1.\text{ver} \leq v_2.\text{ver}$.

- **Agreement**: For any two views $v$ and $w$ returned by getCV()’s, if $v.\text{ver} = w.\text{ver}$, then $v.\text{set} = w.\text{set}$.

- **Eventual Convergence**: If membership pattern $\Pi$ is eventually stable, then there is a view $v$ such that $v.\text{set} = \text{sset}(\Pi)$ and for any node $x \in \text{sset}(\Pi)$, there is a time $t_1$ such that if $x$ invokes getCV() after time $t_1$, the return value is $v$.

- Can be implemented using consensus and eventually perfect failure detector
Weakly Consistent KBR (W-KBR)

- Node id: $x.id \in \mathcal{K}$
- Lookup primitive:
  - $w$-lookup$(k)$, $k \in \mathcal{K}$
  - returns $x$, including $x.id$ and $x.address$, or return $\bot$
W-KBR properties (consistency related)

- **Eventual Progress**: If membership pattern \( \Pi \) is eventually stable, then there is a time \( t_1 \) such that for any key \( k \in \mathcal{K} \), if a node \( x \in \text{sset}(\Pi) \) invokes \( \text{w-lookup}(k) \) after \( t_1 \), then the return value must be some \( y \in \text{sset}(\Pi) \).

- **Eventual Consistency**: If membership pattern \( \Pi \) is eventually stable, then there is a time \( t_1 \) such that for any key \( k \in \mathcal{K} \), if two nodes \( x_1, x_2 \in \text{sset}(\Pi) \) invoke \( \text{w-lookup}(k) \) after time \( t_1 \) and the return values are \( y_1, y_2 \in \Sigma \) respectively, then \( y_1 = y_2 \).
How to achieve W-KBR?

- Many existing protocols are close but no formal analysis.
  - Chord and some latest self-stabilizing protocols can be proven to achieve W-KBR

- Theoretical result:
  - W-KBR can implement $\Omega$ failure detector
    - $\Rightarrow$ W-KBR can implement Consensus
    - $\Rightarrow$ W-KBR cannot be achieved in purely asynchronous systems with even one crashes.
    - $\Rightarrow$ We need some synchrony assumption
  - We assume eventually synchronous and fully connected links
    - Studying minimum synchrony assumption is a future work
Strongly Consistent KBR (S-KBR)

- Intuitively, routings with the same key always return the same answer.
- But the system is changing.
- Augment with key version number.
- Primitive:
  - \( s\text{-lookup}(k), k \in \mathcal{K} \)
  - returns \((x, k\text{ver}), \) or return \(\bot\)
S-KBR properties

- **Causality Consistency:** If two nodes $x_1$ and $x_2$ invoke $s$-lookup($k$) and get return values $(y_1, kver_1)$ and $(y_2, kver_2)$ respectively, and the return of $x_1$’s invocation causally precedes the $x_2$’s invocation, then $kver_1 \leq kver_2$.

- **Strong Consistency:** If two nodes $x_1$ and $x_2$ invoke $s$-lookup($k$) and receive return values $(y_1, kver_1)$ and $(y_2, kver_2)$ respectively, and $kver_1 = kver_2$, then $y_1 = y_2$.

- **Eventual Stability:** If membership pattern $\Pi$ is eventually stable, then there is a time $t_1$ such that for every $k \in \mathcal{K}$, there is a version number $m_k$, for every node $x \in sset(\Pi)$, if $x$ invokes $s$-lookup($k$) after time $t_1$, the return values must be non-$\bot$, and the version number in the return value is $m_k$. 

Feb. 28, 2006
Zone-Based S-KBR Algorithm

- Peers are partitioned into zones (static)
- Each zone maintained by strong group membership service (GMS)
- Routing between zones use W-KBR (a variant)
- Once routed into the target zone, select a node based on the key and the view, and $kver$ is the version of the view.
- If W-KBR fails to reach the target zone, S-KBR fails
Key Points of S-KBR Algorithm

- Zone partition to increase scalability
- W-KBR for locating the zone, and return failure if not found
  - Tradeoff between progress and consistency
- Provably correct
Proposals for Scalability and Adaptability

Zone size determination

- Larger zones, more cost and more consistency and vice versa ⇒ continuum of consistency levels
- Need analysis and simulation support
Proposals (cont’d)

• Zone merge/split
  • Zone merge: when the number of nodes in a zone is too low
  • Zone split: when the number of nodes in a zone is too high

• Changes required
  • Zone version maintenance
  • Zone merge requires inter-zone agreement ⇒ require higher level consistency
Dead zone removal and reactivation

- A zone becomes dead if a majority of nodes crash or leave the system
- Dead zone cannot make progress by itself
- Requires zone monitoring and reactivation by neighboring zones ⇒ inter-zone agreement and higher level consistency
Ongoing and Future Work

- Hierarchical design
  - Systematic approach for zone merge/splits and dead-zone handling
  - Higher level only deals with inter-zone level changes ⇒ rare invocation, not on critical paths for routing and normal maintenance
  - Applicable to the maintenance of large-scale and dynamic systems
  - Need correctness proof
  - Plan to implement in WiDS and verify by simulations
Ongoing and Future Work (cont’d)

- Weakening network model
  - Do not need eventually synchronous and fully connected links
  - Similar to existing work related to Ω failure detector
  - Apply and adjust to P2P context
Questions?