Fault-Tolerant P2P Crawling of Rich Internet Applications

Khaled Ben Hafaiedh, Gregor v. Bochmann, Guy-Vincent Jourdan, Iosif Viorel Onut

School of Electrical Engineering and Computer Science - University of Ottawa

SOFTWARE SECURITY RESEARCH GROUP

In Collaboration With IBM

uOttawa

Introduction – Traditional vs. Rich Internet Applications

Traditional Web Applications

- Sending a request for a URL from the client to the server so that the corresponding web page is downloaded in response for each URL request.
- Each web page is identified by its URL and has only a single state.

Rich Internet Applications

- Interactive and more responsive applications, referred to as RIAs.
- RIAs combine the client-side scripting with new features such as AJAX (Asynchronous JavaScript and XML).
- JavaScript functions allow the client to modify the currently displayed page, by communicating with the server asynchronously.

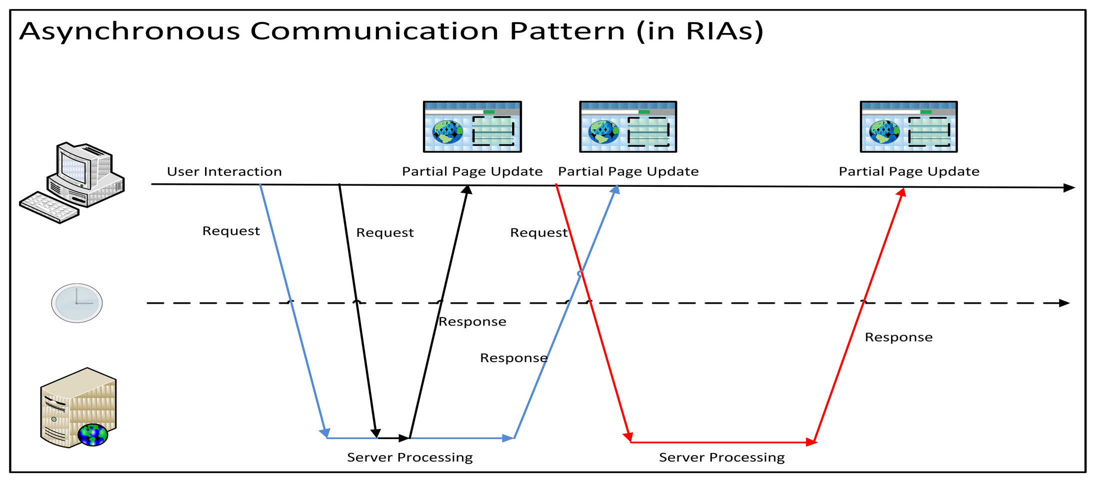

Figure 1. Asynchronous Communication Pattern in RIAs

RIA Crawling

The purpose of a RIA crawler is to automatically exploring all states of a a RIA

Goal

- Context indexing
- Testing for security
- Building application models

Distributed RIA Crawling

Aim

- Running multiple crawlers to reduce the crawling time.
- Sharing the searching space in a single storage unit, called the controller.
- The controller tells the crawlers what to do next.

Challenges

- **Scalability:** The controller may become a bottleneck when it is accessed simultaneously by a high number of crawlers.
- **Fault-Tolerance:** A failure occurring within this unit may result in the entire loss of the graph under exploration.

Motivation

- **Scalability:** A scalable system composed of multiple controllers where a high number of crawlers may be associated with each controller, without having a central bottleneck.
- **Fault-Tolerance:** The crawling system must achieve the crawling task properly when both crawlers and controllers are vulnerable to node failures.

Architecture

- A P2P crawling system composed of multiple controllers [1].
- States are partitioned into disjoint sets and each set is assigned to a particular controller.
- Each controller is associated with a certain number of crawlers responsible of executing events.

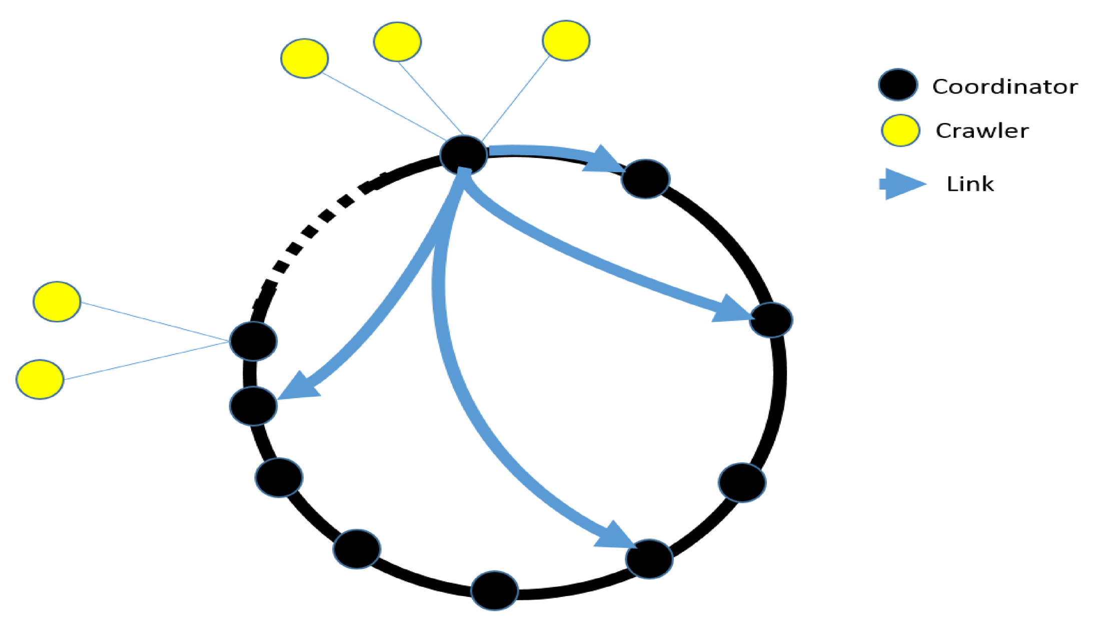

Figure 2. Distribution of states and crawlers among controllers: Each state is associated with one controller, and each crawler gets access to all controllers through a single controller it is associated with.

Assumptions

- Crawlers and controllers are vulnerable to Fail-stop failures, i.e. they may fail but without causing harm to the system.
- Perfect failure detection and reliable message delivery: This allows nodes to correctly decide whether another node has crashed or not.
- Controllers must be reliable as they are responsible for storing information about the RIA crawling.
- Crawlers can be unreliable as they do not store any relevant information about the state of the RIA.

Fault Tolerant P2P RIA Crawling

Crawlers and controllers must achieve two goals in parallel:

1. **Maintaining Chord.**
2. **Recovering lost states and transitions** using a Data-Recovery Mechanism when a failing controller is detected.

1. Chord Maintenance

- The maintenance of Chord consists of maintaining its topology as controllers join and leave the network and repairing Chord independently of the RIA crawling.
- A repair protocol [2] runs periodically by every single controller where each controller attempts to update its routing information.

2. Data-Recovery Mechanisms

- **Retry Strategy:** Replaying any erroneous task execution, hoping that the same failure will not occur in subsequent retries, i.e. re-executing all lost transitions a failing controller was responsible for.
- **Redundancy Strategy:** Maintaining back-up copies of the set of states that are associated with each controller, along with the set of transitions on each of these states and their status, on the successors of each controller.
- **Combined Strategy:** Periodically copying the executed transitions a controller maintains so that if the controller fails, a portion of the executed transitions remains available to the back-up controller, and the lost transitions that have not been copied have to be re-executed again.

Fault Tolerant P2P RIA Crawling

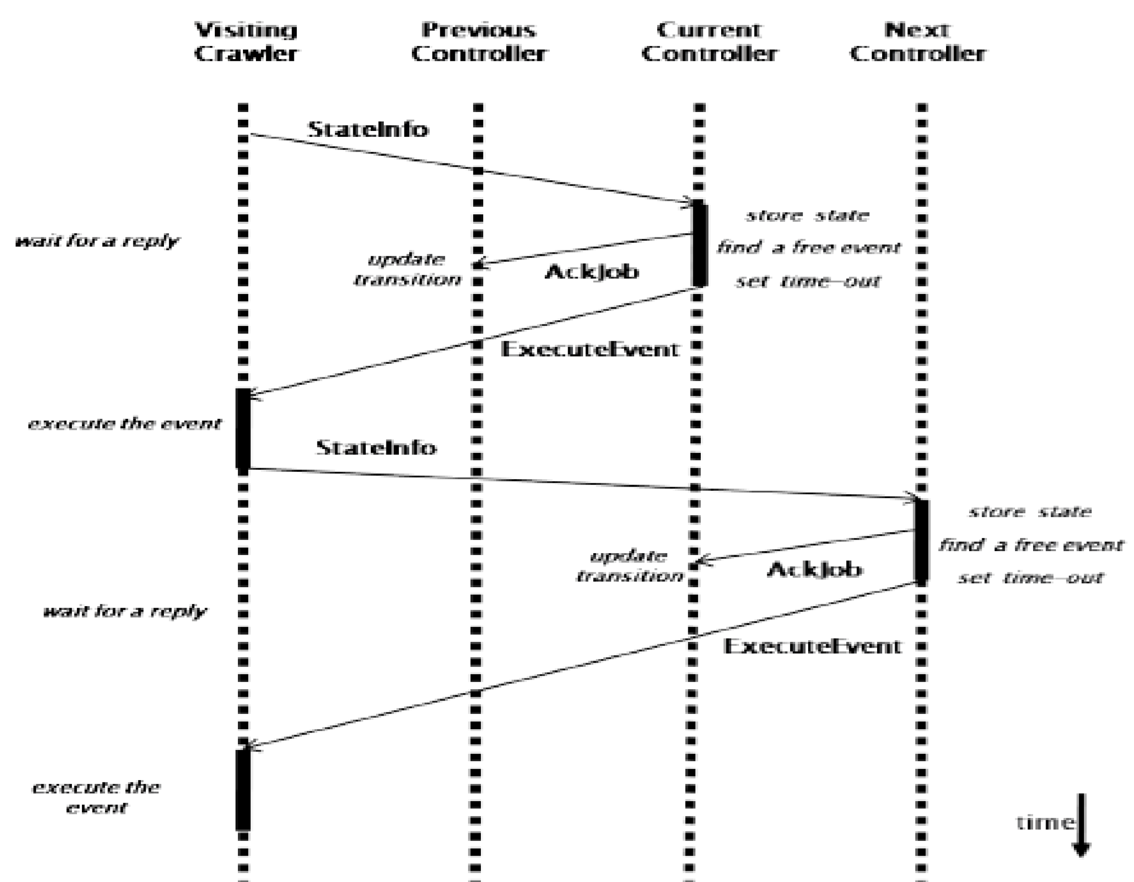

Figure 3 The Fault Tolerant P2P RIA Crawling during the exploration phase.

1. The crawler searches for the controller associated with a state when a new state is reached, by sending a *StateInfo* search message.
2. The controller returns in response a new transition to be executed by sending an *ExecuteEvent* message.
3. The controller sets a Time-out to the assigned transition. When the Time-out expires, the transition is reassigned to a another crawler at a later time.
4. The crawler executes the assigned transition, by either returning to the initial state and retracing the steps that lead to a state with an un-executed event (Reset), or by executing a path of transitions to reach a state with an un-executed event without performing a Reset.
5. The crawler forwards the information about the newly reached state by sending a *StateInfo* message to the next controller.
6. The crawler sends the result of the execution back to the previous controller (*AckJob* message).
7. Upon receiving an *AckJob* message, the previous controller updates the destination state of the transition.

Theoretical Analysis

Parameters:

- Failure rate of the P2P crawling system: $\lambda_f = 1$ failure per hour (Given)
- Communication delay between two nodes: $c = 1$ millisecond (Measured)
- Number of controllers: n (Given)
- Time required for executing one transition: t_t (measured)
- Update Period: T_p (Calculated)
- Processing time for updating the database (Consecutive updates): p (Measured)
- Number of transitions in a RIA: k (Given)

Case when Controllers are Under-loaded

$$Overhead_{Retry} = \frac{\lambda_f . T_p}{2}, \quad where \quad T_p = \frac{k.t_t}{n}$$

$$Overhead_{Redundancy} = \frac{2.c}{T_p} + \frac{p}{t_t}, \quad where \quad T_p = t_t, \quad p \quad depends \quad on \quad \frac{T_p}{t_t}$$

- Measurement of the processing time **p** for updating the database

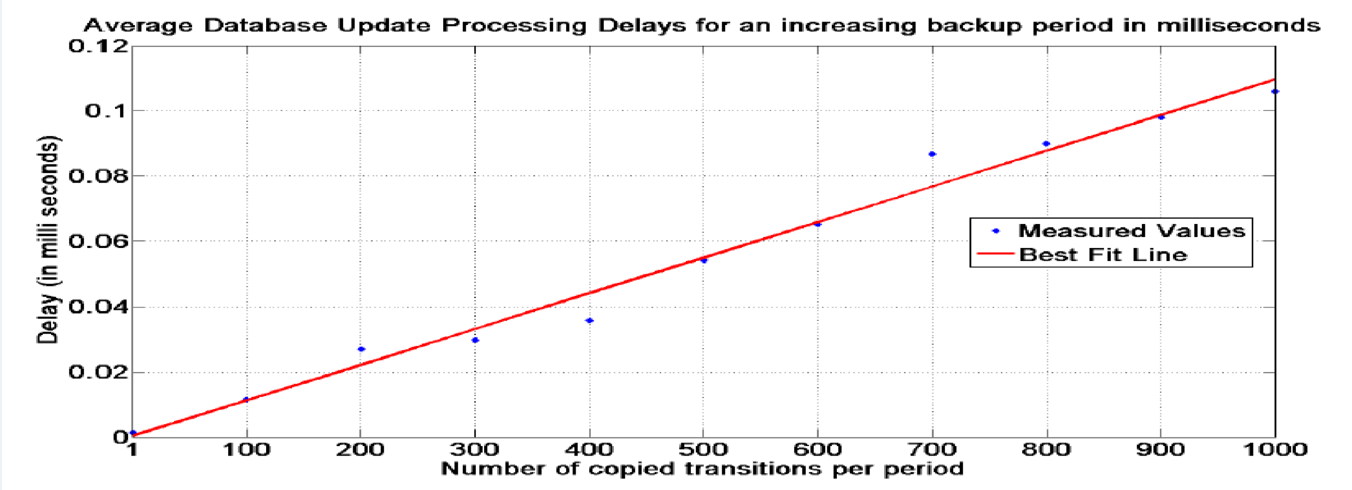

Figure 4. p corresponds to the slope of the line

- Comparison of the Retry and the Redundancy Strategies

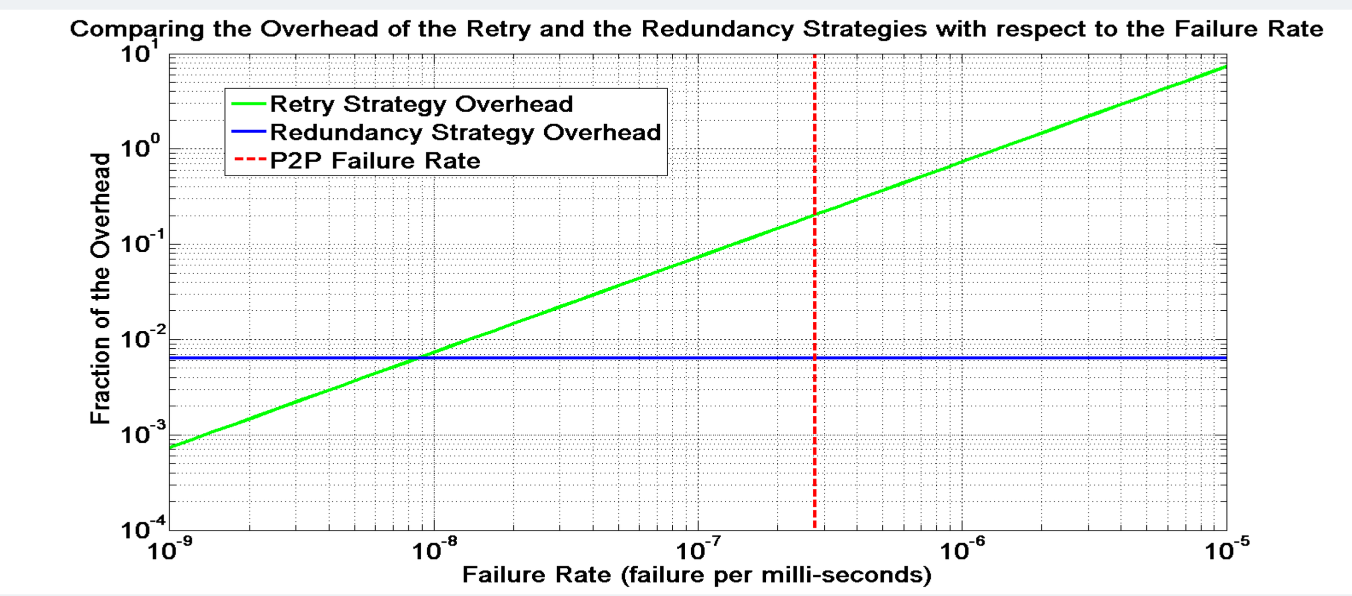

Figure 5. Comparing the Retry and the Redundancy Overheads

Case when Controllers are Over-loaded

- Controllers may become a bottleneck since an update is required for each newly executed transition using the Redundancy Strategy.
- Solution: Periodically copying the executed transitions a controller maintains and re-executing transitions that have not been copied (Combined Strategy)

$$Overhead_{Combined} = \frac{\lambda_f . T_p}{2} + \frac{2.c}{T_p} + \frac{p}{t_t}, \quad where \quad t_t \leq T_p \leq \frac{k.t_t}{n}$$

- What is the value of T_p with minimum overhead using the Combined Strategy ?

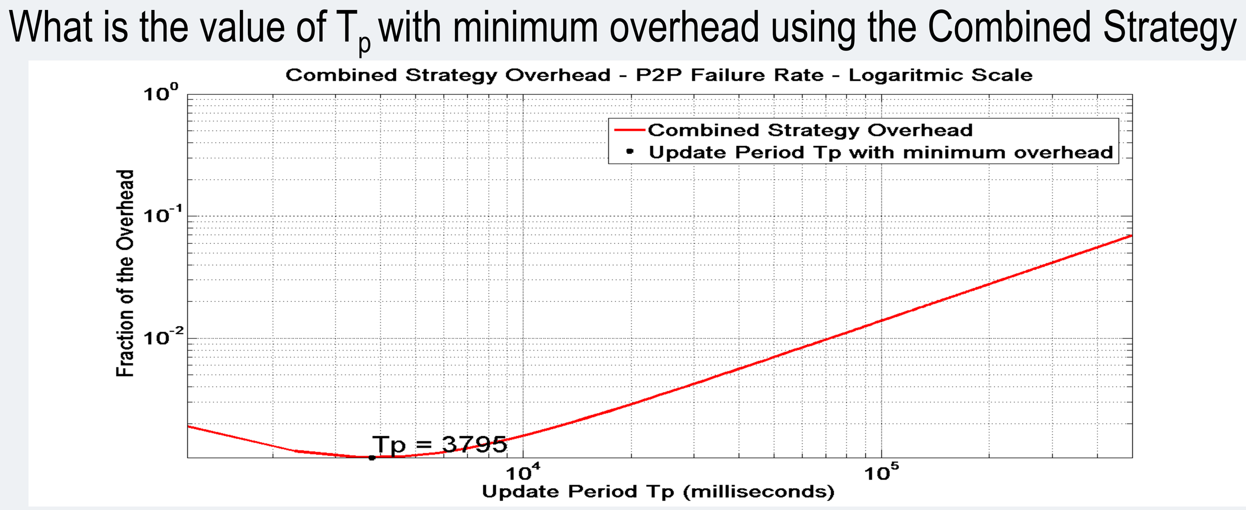

Figure 6. Calculation of the Update Period T_p with minimum overhead

Conclusion & Future Work

- The theoretical analysis shows that:
  1. The Redundancy Strategy is more efficient than the Retry strategy when the controllers are under-loaded.
  2. The Combined Strategy is more efficient than the Retry and the Redundancy strategies when the controllers are over-loaded.
- Future Work: Evaluating the impact of the Combined strategy on the crawling performance when controllers concurrently perform updates.

References

1. Ben Hafaiedh, K., Von Bochmann, G., Jourdan, G. V., Onut, I. V. : A Scalable Peer-to-Peer RIA Crawling System with Partial Knowledge. In: proceedings of the International Conference on Networked Systems, pp. 185--199, Marrakech, Morocco, (2014)
2. Li, X., Misra, J., Plaxton, C. G.: Concurrent maintenance of rings. In: proceedings of the 23rd ACM Symposium on Principles of Distributed Computing, University of Texas at Austin, pp. 376--376, (2004)