SOFTWARE SECURITY RESEARCH GROUP
In Collaboration With IBM

Distributed Large-Scale 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

Introduction – Traditional vs. Rich Internet Applications

Traditional Web Applications

- The typical interaction between the client and the server in a traditional web application consists of 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

- Modern web technologies gave birth to 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.

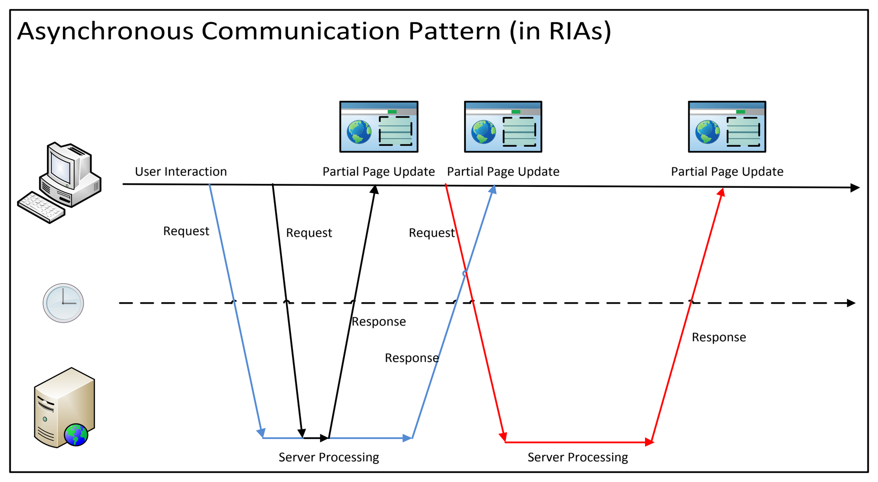


Figure 1. Asynchronous Communication Pattern in RIAs

RIA Crawling

The purpose of a RIA crawler is to automatically exploring states of a rich internet application.

Goal

- Context indexing
- Testing for security
- Building application models

Distributed RIA Crawling

Aim

- Due to the large size of RIAs and therefore the high CPU usage required for the crawl, distributed centralized crawling has been introduced to reduce the amount of time required to crawl RIAs.
- It consists of running multiple crawlers simultaneously and sharing the searching space in a single storage unit, called the coordinator.

Challenges

- **Scalability:** The coordinator 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

We introduce a distributed peer-to-peer architecture for crawling RIAs composed of multiple controllers, referred to as state-holders. Each state-holder maintains a partial model of the application, where a high number of crawlers may be associated with each state-holder. The following contributions are considered:

- **Fault-Tolerance:** The distribution of responsibilities for the states among multiple state-holders in the underlying P2P network, where each state-holder maintains a portion of the application model, so that there is no single point of failure.
- **Scalability:** A scalable system where a high number of crawlers may be associated with each state-holder, without having a central bottleneck that may result from a single database simultaneously accessed by all crawlers.
- **Load balancing:** The balance of the workload among crawlers.
- **Knowledge sharing:** Defining and comparing different knowledge sharing schemes for efficiently crawling RIAs.

Architecture

- A peer-to-peer crawl system composed of multiple state-holders in the form of a chordal ring.
- DOM states are partitioned into disjoint sets, each of which is handled by a distinct state-holder.
- Each state-holder is associated with a certain number of crawlers responsible of executing events.
- The chordal ring allows for:
  - Boosting look-up queries in the ring
  - knowledge sharing among state-holders during the crawl using skip links.

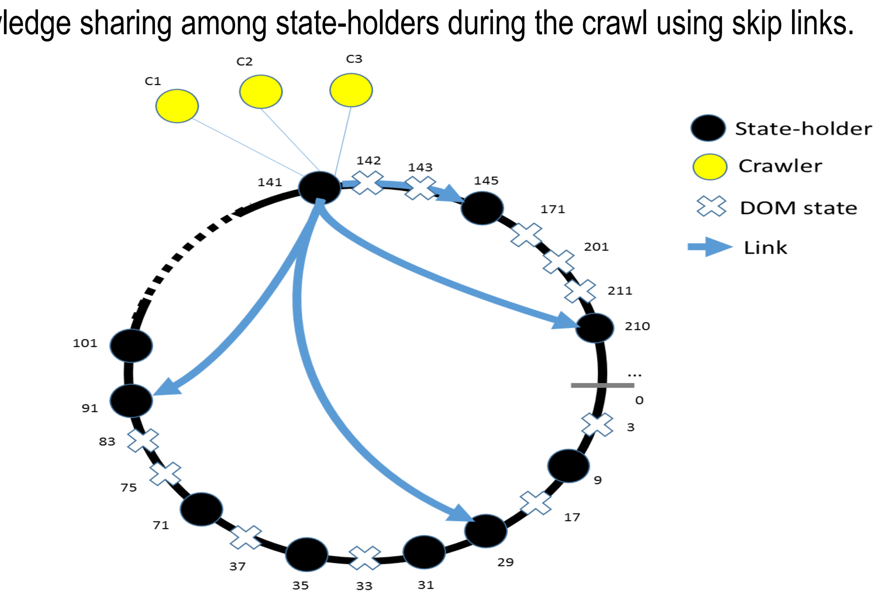


Figure 2. Distribution of DOM states among state-holders: Every DOM state is associated with a state-holder with the largest previous ID

Assumptions

The state-holder

- State-holders do not know the number of state-holder in the network.
- Each state-holder maintains a unique identifier, which is used to distinguish among distinct state-holders in the peer-to-peer network.
- Each DOM state has a unique identifier, which is used to identify the position of the state-holder that is responsible for it in the peer-to-peer system.

The crawler

- Each crawler maintains a unique identifier, allowing a state-holder to distinguish among crawlers that are associated with it.
- Each crawler may locally maintain a copy of the RIA application, allowing for :
  - The speed-up of crawling large-scale applications.
  - Avoiding a central bottleneck resulting from a single RIA application simultaneously accessed by multiple crawlers.

Methodology

The greedy strategy

- The basic greedy strategy consists of exploring an event from the current state if there is any unexplored event. Otherwise, the crawler may execute an unexplored event from a different state, until all transitions are traversed.
- In the centralized crawl system, each crawler may retrieve the required graph information by communicating with the single coordinator, and then executes a single unexecuted event from its current state if such an event exists, or may move to another state with some unexecuted events based on the information available on the single database.
- In the P2P environment, the state-holder responsible of leveraging access to a given state is contacted by each crawler to execute a single unexecuted event on this state. Upon executing an event, a crawler may reach a new state and thus communicate with its corresponding state-holder requesting for a new event execution.
- If no event can be executed on the current state of the crawler, the crawler may communicate with other state-holders to execute events from another state.

Challenges

- **Efficiency:** Crawlers must efficiently execute the graph transitions by only communicating with as few state-holders as possible.
- **Termination detection:** An idle crawler cannot know a priori if all transitions on all states that are maintained by different state-holders have been already executed or not.

Choosing the next event to explore from a different state

Four approaches for finding and executing events on a state other that the current state of the crawler in the P2P crawl system.

Reset-Only

- A visited state-holder picks any other state with an unexecuted event, and returns an execution path with ordered transitions, starting from the initial state, forcing the visiting crawler to perform a reset and to traverse this path before reaching the target state.
- Reset-Only is the simplest way for distributively crawling RIAs. However, this approach results in a high number of resets performed, which may increase the time required to crawl a given application (cost).

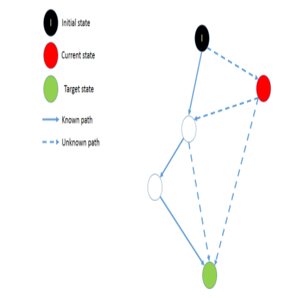


Figure 3. Possible path in the Reset-Only approach

Shortest Path based on local knowledge

- The shortest-path algorithm makes optimal choice for a crawler by finding the shortest path from its current state to any state with an unexecuted event the state-holder is responsible for, without necessarily performing a reset.
- A visited state-holder may use its local transitions knowledge to find the shortest path from the crawler current state leading to the closest unexecuted event.
- The number of known transitions to a state-holder remain relatively low comparing to all executed graph transitions in the RIA model.

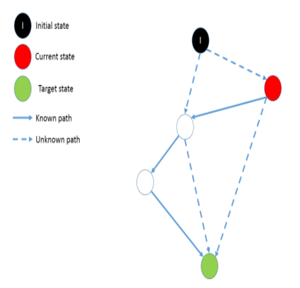


Figure 4. Possible path in the shortest path based on local knowledge approach

Shortest Path based on large knowledge

- Whenever a crawler sends a message to a destination state-holder, all forwarding state-holders in the chordal ring, i.e. intermediate state-holders receiving a message that is not designated to them, may also update their transitions knowledge before forwarding it to the next state-holder in the chordal ring.
- The transitions knowledge is significantly increased among state-holders with no message overhead, allowing state-holders to have more knowledge about the executed transitions, and therefore to reduce the sizes of the computed shortest paths.

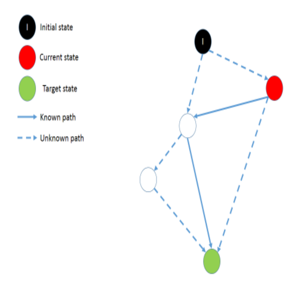


Figure 5. Possible path in the shortest path based on large knowledge approach

Forward-Exploration

- The Forward-Exploration approach is based on the breadth-first search and consists of sequentially moving to the neighboring states from the current state of a visiting crawler.
- In order to prevent different state-holders from visiting states that have already been visited and has no unexecuted events, state-holders may share during the forward exploration their knowledge about the visited states with no unexecuted transitions, along with all executed transitions on these states, with other state-holders in the network.
- This allows for preventing the states with no unexecuted event that have been already seen, from getting visited again.

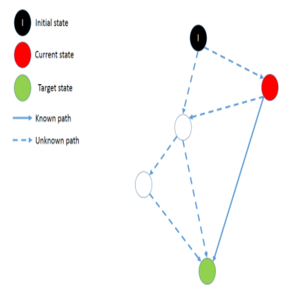
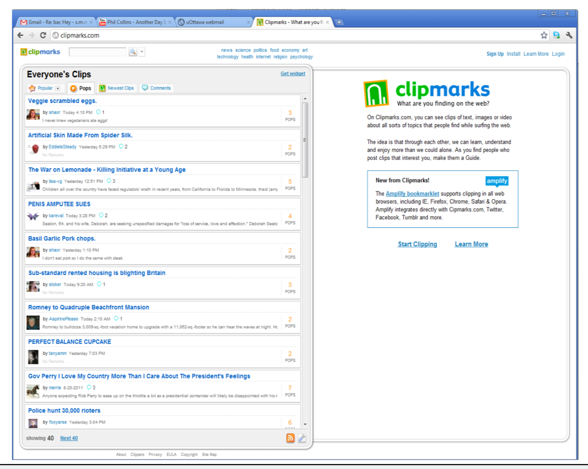


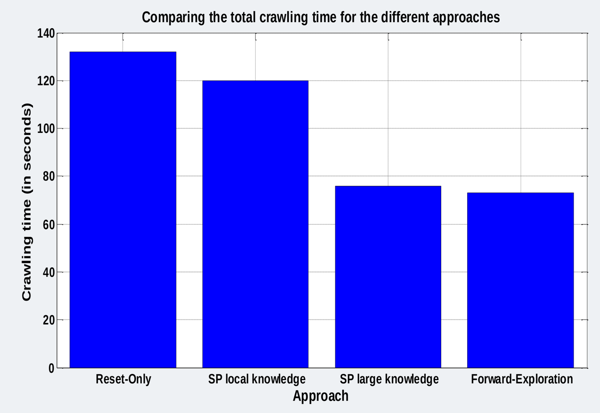
Figure 6. Possible path in the Forward-Exploration approach

Results

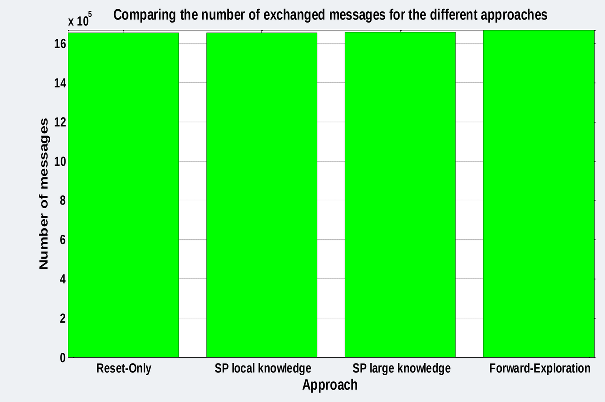
- Tested application: http://clipmarks.com
  - Total number of states: 2663
  - Total number of transitions: 355201
  - Average number of events per state: 133
- Simulation settings
  - Average communication delay: 1 ms
  - Average event execution delay: 11 ms
  - Average number of crawlers a state-holder could handle: 100



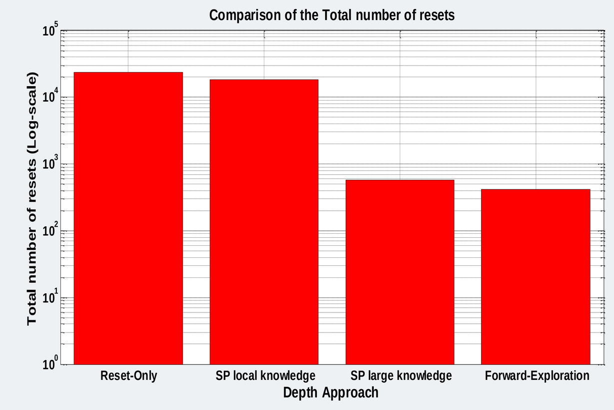
- Comparing the different approaches for 5 state-holders and 100 crawler



- In terms of crawling time, the Forward-Exploration approach is significantly more efficient than the Reset-Only and the shortest path with local knowledge approach, and it slightly outperforms the shortest path with large knowledge.



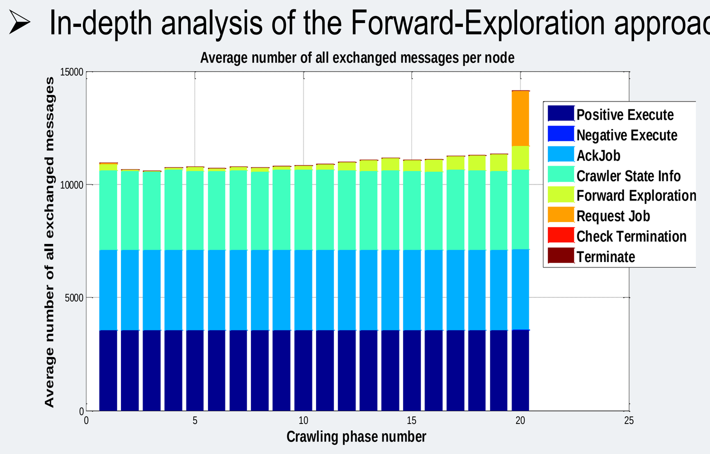
- The total number of exchanged messages is comparable for the Reset-Only, the shortest path with local knowledge and the shortest path with large knowledge.
- The total number of exchanged message with the Forward-Exploration approach is a little higher than the one for all the other approaches.
- This is due to the distributed breadth-first search that is performed by the Forward-Exploration approach when choosing the next event to explore from a different state.



- The number of resets performed consists of the number of times a crawler has to come back to the initial state before reaching a target state (reset cost).
- The shortest path with local knowledge approach slightly outperforms the Reset-Only approach.
- The shortest path with large knowledge approach is significantly better than both the Reset-Only and the shortest path with large knowledge approaches.
- The minimal number of resets performed is obtained with the Forward-Exploration approach.

Figure 7. Comparing the different approaches for 5 state-holders and 100 crawler

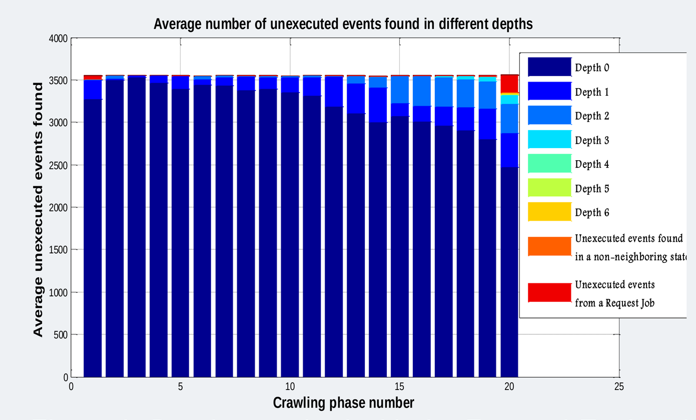
- In-depth analysis of the Forward-Exploration approach: Exchanged messages



- The Forward-Exploration message is only used by the Forward-Exploration approach, while other messages are used by all approaches.
- A Positive Execute message corresponds to a message execution from a state-holder to a crawler with an event to be executed, while a Negative Execute message has no event to be executed.
- Request Job messages are sent by an idle crawler that is looking for a job, by asking other state-holders.
- A high number of Request Job messages are sent during the last crawling phase (before reaching the termination).

Figure 8. Exchanged messages with the Forward-Exploration approach for 5 state-holders and 100 crawlers

- In-depth analysis of the Forward-Exploration approach: Unexecuted events found in different depths during the Forward Exploration operation



- Each depth corresponds to the distance of an unexecuted event found in a neighboring state from a crawler current state.
- The unexecuted events found in a non-neighboring state are unexecuted events that cannot be reached by the forward exploration.
- An unexecuted event found from a Request Job message corresponds to an assigned event to an idle crawler.
- Most of the unexecuted events are found in lower depths and thus are close to the crawler current state.
- The highest depths are reached as we approach the end of the crawl.

Figure 9. Depths reached with the Forward-Exploration for 5 state-holders and 100 crawlers

Conclusion & Future Work

- The experimental results show that the Forward-Exploration approach is more efficient than the Reset-Only, the shortest path with local knowledge and the shortest path with large knowledge.
- Some of the areas that we are currently working on are:
  - Fault-tolerance.
  - Applying other crawling strategies such as the menu model, the component-based model and the probabilistic strategy.

Acknowledgments

This work is supported in part by IBM and the Natural Science and Engineering Research Council of Canada.

DISCLAIMER
The views expressed in this poster are the sole responsibility of the authors and do not necessarily reflect those of the Center for Advanced Studies of IBM.

www.PosterPresentations.com