Dist-RIA Crawler: A Distributed Crawler for Rich Internet Applications

Seyed M. Mirtaheri, Di Zou, **Gregor v. Bochmann**, Guy-Vincent Jourdan, and Iosif Viorel Onut

8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)

October 28, 2013
Overview

• Introduction
  ▫ The evolution of the Web Crawling
• Crawling RIAs
  ▫ Challenges and Assumptions
  ▫ Parallel Crawling
• Our Approach – Distributed Crawling
  ▫ Overview
  ▫ Partitioning Algorithm
• Experimental Results
• Future Work
Introduction - Rich Internet Applications

- RIAs shift parts of the computation to the client
- Client side code changes the “page” - the Document Object Model (DOM).
- Events: Occurrences that cause code execution (mouse click, timeout etc.)
Introduction - Crawling

• Crawling: Automatic exploration of the application

• Motivations
  ▫ Content indexing (by search engines)
  ▫ Testing (for security and accessibility)

• Objectives
  ▫ Find all (or ‘important’) pages
  ▫ Find connections between the pages (page ranking and obtaining a complete model of the application)

• Crawling extracts “a model” of the application
  ▫ States are the “distinct” pages
  ▫ Transitions are the connections between the states
Introduction - chronology of web crawling:

1. Traditional crawling:
   - Every URL is mapped to a single state
   - Crawling is finding all URLs

2. Deep-web crawling:
   - HTML forms are used to access data
   - Crawling involves assigning values to open fields

3. RIA web crawling:
   - Client side events are used to modify the DOM
   - Crawling involves executing all events in each state
Crawling RIAs - Challenges

• State Identification
  ▫ DOM equivalence
  ▫ Detecting independent widgets

• Event Identification

• Deterministic Behaviour (No Server-side States)

• Intermediate States

• Performance

• Our Focus: Performance
Crawling RIAs - Parallel Crawling

• Parallel crawling in traditional Web Apps
  ▫ WebCrawler & MOMspider
    • First parallel crawlers
  ▫ Google
    • PageRank, Compression, etc
  ▫ Mercator, Polybot, pSearch, IRLbotpages
    • Focus on URL-Seen

• Very little work on Parallel Crawling of RIAs:
  ▫ 2011, Mesbah et al.
    • Multi-threading, shared memory
Our Approach - Overview

• Executing events is time consuming
• We (statically) distribute the responsibility of executing client-side events among nodes
• Together, all nodes execute all events on all states and discover the entire model of the application.
Our Approach - Partitioning algorithm

• Each node goes to every state. However, each node executes only some of the events.
• The partitioning algorithm decides which are the events that are executed by each node.
  ▫ All events should be executed
  ▫ No work duplications
• Examples:
  ▫ Range based partitioning
  ▫ Stride based partitioning
  ▫ Hash based partitioning
**Our Approach - Crawling Algorithm**

```
GETCREDENTIALSFROMCOORDINATOR()
NODESTATUS ← ACTIVE
while (NODESTATUS is not TERMINATED) do
    if WORKINGSTATES is Empty then
        GETNEWSTATESFROMCOORDINATOR()
        if WORKINGSTATES is Empty then
            NODESTATUS ← DONE
            SENDINGNODESTATUSTOCOORDINATOR()
        else
            NODESTATUS ← ACTIVE
        end if
    else
        stateToVisit ← PICKSTATE(WORKINGSTATES)
        eventToExecute ← PICKUNEXECUTEDEVENT(stateToVisit)
        EXECUTEEVENT(stateToVisit, eventToExecute)
        if CURRENTSTATE is not in DISCOVEREDSTATES then
            push CURRENTSTATE to DISCOVEREDSTATES
            push ASSIGN(CURRENTSTATE) to CURRENTSTATE.UNEXECUTEDEVENTS
            push CURRENTSTATE to WORKINGSTATES
            SENDINGNEWSTATETOCOORDINATOR( CURRENTSTATE )
        end if
        stateToVisit.REMOVEUNEXECUTEDEVENT(eventToExecute)
        if stateToVisit.UNEXECUTEDEVENTS is empty then
            WORKINGSTATES.REMOVESTATE(stateToVisit)
        end if
    end if
end while
```
Experimental Results

• Dist-RIA Crawler: Implemented on a prototype of IBM® Security AppScan®.

• System parameters:
  ▫ Breadth-First crawling strategy
  ▫ Static partitioning algorithm (no dynamic load balancing)
  ▫ Star communication topology
  ▫ 15 nodes + 1 coordinator
Target application

- RIA File Tree browser

Fig. 2: File tree browser RIA screen-shot
Dist-RIA Crawler: Time to crawl two RIAs

Fig. 3: Time to crawl a RIA with multiple nodes: Apache HTTPD source code file browser (left), and Apache Cassandra source code file browser (right).
Dist-RIA Crawler: Idle times
Conclusions

• Performance improvement (essentially) proportional to number of nodes
• Need for dynamic load balancing to avoid idling of nodes (towards the end of a crawl)
• A single coordinator can handle up to approximately 50 nodes
• Ongoing work:
  ▫ Dynamic load balancing
  ▫ Other crawling strategies
  ▫ Other event partitioning algorithms
  ▫ Avoiding single coordinator : Using a P2P setting
References


References


References


Any Questions ??