Building Large Scale Information Systems

REST
Publish/Subscribe
PageRank

Focus of Different Data Models
CAP Theorem

Partition Tolerance
Consistency
Availability

RDBMS
NoSQL (most)

Spring 2013 cs 5301

The evolving database landscape

Non-relational

Marklogic
Vendiant
MecObject
Progress
Objective
Operational

Analytic
Hadoop
Piccolo
Teradata
IBM Netezza
ParAccel
Kognitio
HPC
Horton
Infobright
LucidDB
EMC Greenplum
Calpoly
Actian
VectorWise
MySQL
Vertica

NoSQL

Graph
Enterprise
Acuvu
Neo4j

NOSQL

Cassandra
HBase

Big tables

Mongo
Redis
Couchbase
Cloudbase

Objectivity

Lotus Notes
Stamtamer
InterSystems Cache

Research

Relational

SAP HANA
IBM Informix
Oracle
Persua
IBM DB2
MysqlDB
SAP Sybase ASE

as-a-Service

Salesforce.com
Amazon RDS
Database.com
Postgres Plus Cloud
CloudDB
Google Cloud SQL

New databases

NuoDB
Vertica
NewSQL
MemSQL
InnoDB
SQLFire
Drizzle
Akkuna
Transcend

ScaleBase
ScaleArc
ParElastic
Continuing
Clustering/Chasking

Spring 2013 cs 5301
Background

WEB SERVICES

What is a Web Service?

- Piece of software available over Internet
- Uses standardized (i.e., XML) messaging system
- More general definition: collection of protocols and standards used for exchanging data between applications or systems
Web Service Architecture

- Service-Oriented Architecture

Architecture II

All the technologies are XML based …
Open, Standard Technologies

- XML – tagging data such that it can be exchanged between applications and platforms
- SOAP – messaging protocol for transporting information and instructions between applications (uses XML)

Open, Standard Technologies

- WSDL – a standard method of describing web services and their specific capabilities (XML)
- UDDI – defines XML-based rules for building directories in which companies advertise themselves and their web services
SOAP

- Simple Object Access Protocol
- Format for sending messages over Internet between programs
- XML-based
- Platform and language independent
- Simple and extensible
- Stateless, one-way
  - But applications can create more complex interaction patterns

SOAP Building Blocks

- Envelope (required) – identifies XML document as SOAP message
- Header (optional) – contains header information
- Body (required) – call and response information
- Fault (optional) – errors that occurred while processing message
Simple Example

“Get the price of apples”

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
            <m:Item>Apples</m:Item>
        </m:GetPrice>
    </soap:Body>
</soap:Envelope>

Note: GetPrice and Item are application-specific (not part of SOAP)

WSDL (Web Service Description Language)

- Web services are self-describing
- Description is written in WSDL, an XML-based language through which a web service conveys to applications the methods that the service provides and how those methods are accessed
- WSDL is meant to be read by applications (not humans)
WSDL (Web Service Description Language)

• Standard method of describing Web Services and their capabilities
• Idea: Automate details involved in applications communication
• Operations and messages are described abstractly
• Bound to a concrete network protocol and message format to define an endpoint
• Provide documentation for distributed systems

WSDL Details

• A WSDL document defines services
• Services are collection of network endpoints (ports)
• Abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings
• Allows the reuse of abstract definitions:
  – messages -abstract descriptions of data being exchanged
  – port types -abstract collections of operations
  – concrete protocol and data format specifications for a particular port type constitutes a reusable binding
An example

```xml
<message name="searchSimple1In">
  <part name="program" type="xsd:string"/>
  <part name="database" type="xsd:string"/>
  <part name="query" type="xsd:string"/>
</message>

<message name="searchSimple1Out">
  <part name="Result" type="xsd:string"/>
</message>
```

Example (continued)

```xml
<portType name="Blast">
  <operation name="searchSimple" parameterOrder="program database query">
    <documentation>Execute Blast</documentation>
    <input name="searchSimple1In" message="tns:searchSimple1In"/>
    <output name="searchSimple1Out" message="tns:searchSimple1Out"/>
  </operation>
  ....[other operations]
</portType>
```
Summary

• WSDL document lists functions supported by each web service, inputs and outputs
• To actually call a web service, need interface or tools:
  – SOAP:lite (Perl)
  – Apache Axis (Java)
  – Many others...

UDDI

• UDDI defines an XML-based format that describes electronic capabilities and business processes
• Entries are stored in a UDDI registry
• UDDI Business Registry (UBR)
  – "white pages" – contact info, description
  – "yellow pages" – classification info, details
  – "green pages" – technical data
  – uddi.microsoft.com
REST
Representational State Transfer

Applications on The Web

• In the usual (non-REST) approach we can use the web for applying RPCs
• Dynamic pages are the result of applying remote applications
• It is possible to specify the application and the parameters in the GET request or send them in a POST request
Web Services

• When calling a remote web service:
  – The client sends the method name and the parameters inside an envelope – i.e., wrapped in XML, in the body of a POST request
  – Receives the result wrapped in an XML envelope
  – Uses SOAP

REST

• A design pattern for implementing network systems
• Provides a set of design principles
Resource

• The web is a collection of resources
• A resource has a
  – URI
  – Content
• A resource can be represented in different ways
• A response provides a representation of a resource

• Can we compare two resources by comparing their content?

Different Representations of a Resource

• Consider an HTML page cs5301.html and the following files:
  – The compression of cs5301.html
  – The results of “fixing” cs5301.html to conform to the XHTML standard
  – The presentation of cs5301.html using different CSS stylesheets
  – The file cs5301.html in different character encodings
  – The file cs5301.html in different languages
• Which of the above is the same resource as cs5301.html?
Client Interaction

- The client references a web resource using a URL
- A representation of the resource is returned
- The representation places the client in a new state
- When the client selects a hyperlink it accesses another resource
- The new representation places the client application into yet another state
- Thus, the client application transfers state with each resource representation

Representational State Transfer

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

- Roy Fielding
Services

• In REST, there is a resource for every service
• There is a URL for every resource

• So, how do we call a service?

Application Invocation

RESTful

http://university.edu/students/55456

non-RESTful

http://university.edu/getStudent?id=55456
List of Students

RESTful

http://university.edu/students/

```xml
<?xml version="1.0"?>
<Students>
  <Student id="55345" href="http://www.university.edu/students/55345"/>
  <Student id="55346" href="http://www.university.edu/students/55346"/>
  <Student id="55347" href="http://www.university.edu/students/55347"/>
  <Student id="55348" href="http://www.university.edu/students/55348"/>
</Students>
```

To return a list of resources, it provides a list of URIs

Too Many Addresses?

- Q: If we have 100,000 students, does this mean we need 100,000 web pages? And what if we will want to represent 100 grades for each student?
- A: We just need the method to generate/retrieve the representation of the resource upon request
Operations

• All interactions between a client and a web service are done with simple HTTP operations:
  – Retrieve information (HTTP GET)
  – Create information (HTTP PUT)
  – Update information (HTTP POST)
  – Delete information (HTTP DELETE)

Requirements

• Architecture in REST is required to provide
  – Separation between clients and servers
  – Stateless protocol
  – The ability to cache responses
  – Mediators between clients and servers (e.g., proxies, gateways) should be transparent to users
Benefits of the RESTful Approach

• What are the benefits of using the REST architecture?

• Easier caching:
  – For example, are the following two URIs represent
    
    http://university.edu/getGrade?student=111&course=333
    vs.
    http://university.edu/getGrade?course=333&student=111
    
    http://university.edu/students/111/course/333/grade

  Going back and forth is simply moving from one
  state to another

• What happens in a web shopping application when
  – You put items in the shopping cart
  – Leave the cart untouched (and the browser’s window,
    as well) for a week
  – Tries to continue the purchase

• Is it possible to add as a bookmark a shopping
  “state”?
Benefits of the RESTful Approach

• When a request is sent to an application, on the way to the application, only the headers of HTTP requests are being examined – thus, in REST it is easier to enforce an **access control policy** or distinguish between different requests

• For example, if it is forbidden to access some resources that appear in a blacklist, it is easy to enforce that when the resources are specified in the head of the message (the URL in a RESTful approach) and difficult if they are in the body (a SOAP message)

For More Information


Publish/Subscribe

Outline

• Introduction to Publish/Subscribe Paradigm

• Common Denominators of Pub/Sub Schemes

• How Pub/Sub compares to “Traditional” Interaction Schemes

• Variants of Pub/Sub Schemes

• Design and Implementation
Why Publish/Subscribe?

- A Distributed System - thousands of entities-distributed all over the world-whose location and behavior greatly vary

- There is a demand for a more flexible communication system that reflects the dynamic and decoupled nature of the applications described above

Publish/Subscribe Basics

- Subscribers - have the ability to express their interest in or subscribe to an event or a pattern of events

- Publishers – can publish or advertise events

- Event Notification Service – provides storage and management for subscriptions and efficient delivery of events
Event Notification Service acts as a neutral trusted mediator between publishers and subscribers

Naïve Approach

- Each subscriber will act as a server, listening to notifications
- Each publisher will send notifications to all the clients
- What is the problem?
- Why is this approach different from the publish/subscribe architecture?
Outline

• Introduction to Publish/Subscribe Paradigm
• Common Denominators of Pub/Sub Schemes
• How Pub/Sub compares to “Traditional” Interaction Schemes
• Variants of Pub/Sub Schemes
• Design and Implementation

Space Decoupling

• The interacting parties (publishers and subscribers) do not need to know each other
• Both parties go through the Event Service for all interactions
Time Decoupling

• The interacting parties do not need to be actively participating in the interaction at the same time

Synchronization Decoupling

• Publishers are not blocked while producing events

• Subscribers can get asynchronous notifications of occurrences of events while performing concurrent tasks

• Production and consumption of events does not happen in main flow of control, so interaction does not have to be synchronized
Synchronization Decoupling

Outline

• Introduction to Publish/Subscribe Paradigm
• Common Denominators of Pub/Sub Schemes
• How Pub/Sub compares to “Traditional” Interaction Schemes
• Variants of Pub/Sub Schemes
• Design and Implementation
Alternative Communication Paradigms

- The ‘Cousins’:
  - Message Passing
  - Remote Procedure Call (RPC)
  - Shared Spaces
  - Message Queuing

Message Passing

- Viewed as ancestor of distributed interactions

- Participants communicate by sending and receiving messages asynchronously through a network channel

- Similar to Pub/Sub, but Producer and Consumer are coupled in time and space
Message Passing

- Asynchronous for producer but not consumer
- Channel set up ahead of time, producer and consumer are active at same time
- Recipient of message known to sender

Remote Procedure Call (RPC)

- One of the most widely used forms of distributed interaction
- Remote interactions appear the same as local interactions, make distributed programming easy
- Each request is followed by a response
Remote Procedure Call (RPC)

- Strong time and synchronization (on the consumer side) coupling
- Coupled in space - an invoking object holds a remote reference to each of its invokes

Attempts to remove synchronization in RPC

**Asynchronous RPC**
- Decouples synchronization by making the producer not expect a reply

**Future RPC**
- Decouples synchronization by not blocking the producer, they can access the reply later when it becomes available
Shared Spaces

- JavaSpaces, Tspaces, and Linda

- Distributed shared memory, common to all participants who interact by reading and writing to it

- Many-to-many anonymous interaction – time and space decoupling

- Consumers not asynchronously notified of messages, but retrieve messages with a synchronous request
Message Queuing

- Messages are persistently stored within queues

- All communications filtered by the queue – similar to the Notification Service in Pub/Sub

- Consumers must explicitly pull messages from the queue – No synchronization decoupling on consumer side
Decoupling Abilities of Interaction Paradigms

<table>
<thead>
<tr>
<th>Abstraction</th>
<th>Space decoupling</th>
<th>Time decoupling</th>
<th>Synchronization decoupling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Passing</td>
<td>No</td>
<td>No</td>
<td>Producer-side</td>
</tr>
<tr>
<td>RPC/RMI</td>
<td>No</td>
<td>No</td>
<td>Producer-side</td>
</tr>
<tr>
<td>Asynchronous RPC/RMI</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Future RPC/RMI</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Tuple Spaces</td>
<td>Yes</td>
<td>Yes</td>
<td>Producer-side</td>
</tr>
<tr>
<td>Message queuing</td>
<td>Yes</td>
<td>Yes</td>
<td>Producer-side</td>
</tr>
<tr>
<td><strong>Pub/Sub</strong></td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
</tr>
</tbody>
</table>

Publish/Subscribe

![Diagram](image)
Outline

- Introduction to Publish/Subscribe Paradigm
- Common Denominators of Pub/Sub Schemes
- How Pub/Sub compares to “Traditional” Interaction Schemes
- Variants of Pub/Sub Schemes
- Design and Implementation

Topic-Based Publish/Subscribe

- Earliest pub/sub scheme based on notion of topics or subjects

- Topics similar to notion of groups
  - Subscribing to a topic $T$ can be viewed as becoming a member of a group $T$, and publishing an event on topic $T$ translates to broadcasting event among members of $T$

- Programming abstraction which maps individual topics to distinct communication channels (many-to-many relationship)
Topic-Based Publish/Subscribe

- Hierarchy
  - A subscription made to some node in the hierarchy implicitly involves subscriptions to all the subtopics of that node

- Wildcards
  - Offer the possibility to publish and subscribe to several topics that match a given set of keywords at the same time
    - E.g. an entire subtree or specific level of hierarchy
Topic-Based Example

```java
public class StockQuote implements Serializable {
    public String id, company, trader;
    public float price;
    public int amount;
}

public class StockQuoteSubscriber implements Subscriber {
    public void notify(Object o) {
        if (((StockQuote)o).company == 'TELCO' && ((StockQuote)o).price < 100) {
            buy();
        }
    }
}

// ...
Topic quotes = EventService.connect("/LondonStockMarket/Stock/StockQuotes");
Subscriber sub = new StockQuoteSubscriber();
quotes.subscribe(sub);
```

Content-Based Publish/Subscribe

- Subscriptions are related to specific information content
  - More flexible than Topic-Based

- Consumers subscribe to selective events by specifying filters to define constraints
  - Filters: name-value pairs, or comparison operators
    - (=, <, ≤, >, ≥)
  - Logically combined to form complex subscription patterns

- Each combination of information items can be seen as a single dynamic logical channel (more of a many-to-one relationship)
Content-Based: Subscription Patterns

- Strings
  - Most frequently used
  - Must conform to the subscription grammar
    - SQL, XPath, or some proprietary language
  -Parsed by the engine

- Template object
  - A participant provides an object $t$, which means they are interested in every event that conforms to type $t$ and whose attributes all match the attributes of $t$.

- Executable code
  - Subscribers provide a predicate object able to filter events at runtime.
  - Implementation left to developer, hard to optimize

Content-Based Example

```java
public class StockQuote implements Serializable {
    public String id, company, trader;
    public float price;
    public int amount;
}
public class StockQuoteSubscriber implements Subscriber {
    public void notify(Object o) {
        buy();   // company == ‘TELCO’ and price < 100
    }
}
// ...
String criteria = (“company == ‘TELCO’ and price < 100”);
Subscriber sub = new StockQuoteSubscriber();
EventService.subscribe(sub, criteria);
```

Spring 2013 cs 5301
Type-Based Publish/Subscribe

- Events are filtered by their type
  - Example: Stock events can be split into two distinct types: stock quotes and stock requests

- Reuses type scheme of object-oriented languages, provides seamless integration between middleware and programming language

- Aims at providing guarantees such as type safety and encapsulation

Type-Based Example

```java
class LondonStockMarket implements Serializable {
  public String getId() {...}
}
class Stock extends LondonStockMarket {
  public String getCompany() {...}
  public String getTrader() {...}
  public int getAmount() {...}
}
class StockQuote extends Stock {
  public float getPrice() {...}
}
class StockRequest extends Stock {
  public float getMinPrice() {...}
  public float getMaxPrice() {...}
}
class StockSubscriber implements Subscriber<StockQuote> {
  public void notify(StockQuote s) {
    if (s.getCompany() == 'TELCO' && s.getPrice() < 100)
      buy();
  }
}
// ...
Subscriber<StockQuote> sub = new StockSubscriber();
EventService.subscribe<StockQuote>(sub);
```
Type-Based Example

Advantages
- Efficient implementations since message classification is static
- Routing is simple
- Notifications that don’t match any subscription aren’t sent to clients
- Enables subscribers to describe runtime properties of objects they want
- Simplicity and flexibility of decentralization implementation (large # of clients and data transfers)
- Scalability by the use of content filtering

Disadvantages
- Limited expressiveness
- Inefficient use of bandwidth if subscriber is only interested in specific criteria
- More expressive, but higher runtime overhead
- Complex protocols/implementations to determine the subscriber
- Many events need to be pruned for performance reasons
Outline

• Introduction to Publish/Subscribe Paradigm
• Common Denominators of Pub/Sub Schemes
• How Pub/Sub compares to “Traditional” Interaction Schemes
• Variants of Pub/Sub Schemes
• Design and Implementation

Implementing Topic-Based Pub/Sub

• Using a centralized server (but can be replicated/clumpeded)
  – Requires an efficient mechanism for matching events to subscriptions (this problem resembles documents search and similar techniques can be used for both problems)
  – Main drawback:
    • Servers can become a bottleneck
    • Large fan-out of the event dissemination task, even when applying load balancing across multiple servers
Fan-Out Problem

Using Brokers
Events

• Event Forms:
  
  – Messages – lower level
    • Header (message identifier, issuer, priority, expiration time, etc.) and payload data

  – Invocations - higher level
    • Directed to a specific type of object and has well defined semantics
    • Additional logic to transform low-level messages into invocations to methods of the subscribers

Architecture

(a) Multicasting Services
(b) Broker-level Notification
(c) Peer-to-Peer overlay network
Topic Based Publish/Subscribe

- Option 1: Use a group communication toolkit and define a group for each topic
  - There may be scalability problems if the number of clients is very large, or the number of topics is very large
  - Does not support well multiple topics and range subscriptions
- Option 2: use a fixed overlay of brokers, in which brokers tell their neighbors to which topics they have subscribers
  - Similar to IP multicast
  - For efficiency, use Bloom filters
- Yet, another option is to employ a set of brokers that form a self-organized overlay (P2P)

P2P Implementation of Pub/Sub

- For each topic, build a dissemination tree among the brokers that have active subscribers
  - All events will be flooded along the tree arcs
  - In this case, a broker that receives a new subscription should join as a leaf
  - A broker that no longer has active subscriptions, should leave the tree
P2P Implementation of Pub/Sub

- Questions:
  - How to build this tree dynamically?
  - We would like to load balance the job of being the root for different topics (trees)
  - We would like to ensure that most brokers that have no active subscribers for a given topic will not be included in the tree

- Possible solution:
  - Utilize a DHT

Self-Organization of Brokers for Publish/Subscribe

- Subscribing protocol
  - A subscriber contacts its nearest broker and passes the subscription to the broker
  - A broker $b$ that receives a subscription:
    - Creates an entry (e.g., in a hash table) with a back link to the node (broker or client) that sent it the subscription
    - Uses a hash of the subscription tag to ask the DHT for the next node $n$ to route to
    - If $n \neq b$, then $b$ forwards the subscription message to the next node $n$

- Unsubscribing protocol
  - The opposite
Publishing Protocol

- A publisher contacts its nearest broker and passes to it the tagged event in an UP message
- A broker $b$ that receives a tagged event in an UP message, asks the DHT for the next node $n$ to route to
  - If $n \neq b$, then $b$ forwards the UP message to the next node $n$
  - Otherwise, $b$ sends the tagged event in a DOWN message on every back link that corresponds to a subscription it knows about that matches the event’s topic

Publishing Protocol

- A broker $b$ that receives a tagged event in a DOWN message
  - Sends the tagged event in a DOWN message on every back link that corresponds to a subscription that matches the event’s topic
- When a subscriber gets a tagged event that matches any of its subscriptions, it passes the tagged event to the application
Qualities of Service

- Persistence
- Priorities
- Transactions
- Reliability

Information-driven Applications

- Communication is indirect and initiated by publishers of information
- News delivery
- Stock quoting
- Air traffic control
- E-commerce
- Social networking
- Anomaly detection
- Electronic mailing lists or bulletin boards
- Chat room instant message services
Advantages/Disadvantages of Pub/Sub

• Advantages
  – Scalability
  – Loosely coupled

• Disadvantages
  – Loosely decoupled

References

• The Many Faces of Publish/Subscribe

• Publish/Subscribe Communication Systems: from Models to Application

• Survey of Publish Subscribe Communication System
  http://medianet.kent.edu/surveys/IAD04F-pubsubnet-shennaaz/Survey2.html
PAGE RANK

The search engine spider crawls the web by following hyperlinks on Web pages and compiles a list of pages to be stored in the search engine index.

The search engine's index contains detailed data about each Web page, and what links are associated with each page.

When a user performs a search through the search engine, a sophisticated algorithm is applied to the index, which returns all occurrences of the query in reverse order from most to least relevant.

The Internet is made up of trillion(s) of Web pages that have billions of links between them.
Are all Pages Equally Important?

If a tree falls in the forest, and nobody blogs about it... does it make a difference?

Spring 2013

PageRank
Link Structure of the Web

• The number of links is about 10 times the number of pages

Backlinks and Forward links:
- A and B are C’s backlinks
- C is A and B’s forward link

Intuitively, a webpage is important if it has a lot of backlinks

What if a webpage has only one link off www.yahoo.com?

A Simple Version of PageRank

\[ R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} \]

• u: a web page
• B_u: the set of u’s backlinks
• N_v: the number of forward links of page v
• c: the normalization factor to make \[ \| R \|_1 = 1 \] (\[ \| R \|_1 = | R_1 + ... + R_n | \])
An example of Simplified PageRank

PageRank Calculation: first iteration

\[
\begin{bmatrix}
1/3 \\
1/2 \\
1/6
\end{bmatrix}
= \begin{bmatrix}
1/2 & 1/2 & 0 \\
1/2 & 0 & 1 \\
0 & 1/2 & 0
\end{bmatrix}
\begin{bmatrix}
yahoo \\
Amazon \\
Microsoft
\end{bmatrix}
= \begin{bmatrix}
1/3 \\
1/3 \\
1/3
\end{bmatrix}
\]

PageRank Calculation: second iteration

\[
\begin{bmatrix}
5/12 \\
1/3 \\
1/4
\end{bmatrix}
= \begin{bmatrix}
1/2 & 1/2 & 0 \\
1/2 & 0 & 1 \\
0 & 1/2 & 0
\end{bmatrix}
\begin{bmatrix}
yahoo \\
Amazon \\
Microsoft
\end{bmatrix}
= \begin{bmatrix}
1/3 \\
1/3 \\
1/3
\end{bmatrix}
\]
An Example of Simplified PageRank

Convergence after some iterations

A Problem with Simplified PageRank

A loop:

During each iteration, the loop accumulates rank but never distributes rank to other pages!
An example of the Problem

\[ \begin{bmatrix} 1/3 \\ 1/6 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \]

Spring 2013  cs 5301  102

An example of the Problem

\[ \begin{bmatrix} 1/4 \\ 1/6 \\ 7/12 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/6 \\ 1/2 \end{bmatrix} \]

Spring 2013  cs 5301  103
Random Walks in Graphs

• The Random Surfer Model
  – The standing probability distribution of a random walk on the graph of the web (simply keeps clicking successive links at random)

• The Modified Model
  – The “random surfer” simply keeps clicking successive links at random, but periodically “gets bored” and jumps to a random page based on the distribution of E
Modified Version of PageRank

\[ R'(u) = c_1 \sum_{v \in B_u} \frac{R'(v)}{N_v} + c_2 E(u) \]

E(u): a distribution of ranks of web pages that “users” jump to when they “gets bored” after successive links at random.

An Example of Modified PageRank

\[
M = \begin{bmatrix}
0.5 & 0.5 & 0 \\
0.5 & 0 & 0 \\
0 & 0.5 & 0.5 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
\text{yahoo} \\
\text{Amazon} \\
\text{Microsoft} \\
\end{bmatrix} = \begin{bmatrix}
1/3 \\
1/3 \\
1/3 \\
\end{bmatrix}
\]

\(c_1 = 0.8 \quad c_2 = 0.2\)
Pages with no Outgoing Links

• How should we do with pages with no outgoing links?
  – A self-link
  – Consider then as if they have a link to every page on the graph

• Which approach is better?

Convergence Property

• The Web is an expander-like graph
  – Theory of random walk: a random walk on a graph is said to be rapidly-mixing if it quickly converges to a limiting distribution on the set of nodes in the graph. A random walk is rapidly-mixing on a graph if and only if the graph is an expander graph

  – Expander graph: every subset of nodes S has a neighborhood (set of vertices accessible via outedges emanating from nodes in S) that is larger than some factor \( \alpha \) times of \(|S|\). A graph has a good expansion factor if and only if the largest eigenvalue is sufficiently larger than the second-largest eigenvalue.
Big Data and How it Changes the World

How will Large Scale Information Systems Change our World?

- More privacy or less privacy?
- Greater influence of individuals and small groups or greater influence to rich, strong companies?
- How will the collecting of data change the way people view the world?
Which Grain of Sand Makes a Heap?

- Which bit turns data into Big Data?
- Which piece of information transforms data into knowledge?