Information System Infrastructure II

*Publish/Subscribe*

Hans-Arno Jacobsen
jacobsen@eecg.toronto.edu
www.eecg.toronto.edu/~jacobsen

Motivation
Really cool ..., *but does this fly?*

At COMDEX 2001 in Toronto

- cell phone to access Windows desktop at home
- *e.g.*, download & view Powerpoint presentation quadrant by quadrarnat (no kidding!)
- searching for nearby restaurants on your PDA & phone
- ...

**Push or Pull?**

http://www.yahoo.com/mobile/1241240/82.13%dfd#12/stocks.html

*push!!*
Observations

- a push-style information dissemination model may be more appropriate for mobile devices
- many mobile devices are inherently constraint
  - in terms of data entry
  - in terms of data output & visualization
  - in terms of battery life
- receiving the right data at the right time, without much searching would be desirable

Selective Information Dissemination

2 BR @ King & Bathurst approx. $1100

NASDAQ:
TIBC $32.14
AXS $44.42
...
Remarks

- the selective information dissemination system must know users' interests
- users' interest change over time
- the more detailed users' interests are represented, the more selective information can be provided (...the happier users will be)
- information content must be represented in a way that permits to identify which users the information may concern
- how many users may need to be supported?

How many users must be supported?

From NTT DoCoMo’s www-site.

about 26 M, 7/2002
Quantitatively speaking

- eBay publishes about **560,000 new auction items** per day (1999)
- Advertisement distributor disseminates ads on daily basis to **11 million i-mode** subscribers on behalf of **28 companies** (expected to grow to 700)
- FloNetwork disseminates **540 million e-mails** on behalf of **250 publishers**
- DoubleClick **500 million e-mails** per month for **200 direct marketers**

Remember Me (my query!)

![Google Search Result](image)

Your search – Feuerwehrdrehleiteroperationshandbuch – did not match any documents. No pages were found containing "Feuerwehrdrehleiteroperationshandbuch".

Suggestions:
- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.

(c) 2002 Hans-Arne Jacobsen (jacobsen@eceg.toronto. edu) MIE491f
Querying the Future

- what if the system could remember my query?
- ...and notify me after data becomes available that matches my query?
- ... manage millions of users’ queries ...

... and tell me anywhere ...
- also useful in non-mobile / wireless context

Location-based Services in Mobile Wireless Environments

(user Z likes)

(c) 2002 Hans-Amo Jacobsen (jacobson@eecg.toronto.edu) MIE491f
Observations

- static user interests, as in previous examples
- dynamic user interests
  - let me know if my friend is within the range of 500 meters
  - let me know if there is a traffic problem on my way to work
- static information about the environment, e.g., maps, location of intersections etc.
- dynamic information, such as traffic, weather, and road conditions
- information content, such as special events, special offers in nearby shops & restaurants etc.

Amazon to Chapters to you ...

Monday, October 10th

Saturday, November 3rd

“Web Caching” available at Young & Bloor. $5 off
Observations

- this example is distinguished from the others by the fact that various independent systems need to collaborate to identify the information relevant for the users

How are user interests expressed? (a.k.a. subscriptions)

```
notify, if {  company = "TIBX",
            exchange = "NASDAQ", Price >= $11)
notify, if {  product = "Palm Vx",
            price < 900FF,  location = Paris)
notify, if {  3 successive login--failures occurred }
notify, if {  Toronto / News / Blue Jays
notify, if {  contains word "publish/subscribe" }
notify, if {  message.tag = IBM-stock }
```
How is information content expressed? (a.k.a. publications)

- disk-full, login-failure, ftp-access, port-scan
- \{IBM, (blob)\}, \{Toronto, (blob)\}
- \{ (exchange, NASDAQ), (symbolb, TIBX), (closing price, 78), (currency, USD) \}
- \{ (user, 34252043), (location, Bloor & Young) \}
- ascii-text or unicode text
- html-document (e.g., indexed web-page, google example)
- an xml-document

Overall observations

- large-scale distributed system with interactions
  - de-coupled in time
  - de-coupled in space and location
- large-scale distributed system in terms of
  - large user population & great potential for overlap in users' interest
  - many heterogeneous information resources
- system entities are only loosely coupled
- overall system is highly dynamic
  - entities come up and go down periodically
  - system evolves constantly (new entities come on)
- timely notification requires high frequency polls which can lead to
  - network congestion and overload
  - server resource contention
- de-centralized system management (hard to agree on common schema)
What kind of system model is needed?

- pull-based approach is not well suited for mobile and wireless environments
- pull-based approach is not well suited for the exchange of message among a large number of participants
- due to the high-degree of personalization multicast is not applicable in this context
- addressing individual users’ information requests
- there is a need for
  - efficient matching algorithms
  - push-based enabling technology
  - efficient routing protocols

Conceptually speaking:

- seeking a paradigm to model state change and action for (distributed) applications
- ... a paradigm to model interaction in large-scale distributed systems
- ... a paradigm that supports de-coupled (time, space, location) interaction

This is difficult to achieve via point-to-point request/reply style communication (a.k.a. client-server model).
Also known as the
Publish/Subscribe Paradigm

Overview

I. Overview & introduction
II. Application domains of P/S systems
III. P/S systems & prototypes
IV. Matching algorithm(s)
I. Overview & introduction

Outline
- definition of terminology
- P/S characteristics
- P/S benefits
- other paradigms
- and some exceptions
Publish/Subscribe Systems

- **Publishers**
  - Data source, information provider
- **Subscribers**
  - Data sink, information consumer
  - Specify requirements in subscriptions
- **Brokers**
  - Receive messages from publishers and route them to subscribers whose subscriptions are satisfied

Terminology Event Notification cont.’d

- **Publisher**
  - Declares an intention to send publications
  - Describes publication types and patterns
  - Detects event occurrences
- **Subscriber**
  - Registers interest in receiving publications
  - Specifies subscriptions
- **Notification**
  - Fine-grained messages
  - Signals a match
  - Potential for automatic processing (notification = event)
- **Propagation**
  - Notification servers send notifications to subscribers
  - May have QoS associated with propagation
The publish and subscribe paradigm: Characteristics

- **message-based** communication model
- publisher and subscriber are **de-coupled**
  - in **time** (do not need to be up at the same time)
  - in **space** (do not need to know about each other)
  - in **location** (may be physically distributed)
- i.e., publisher and subscriber are **a priori anonymous**
- often associated with **push-style** communication (however, many P/S systems support push & pull modes)

Benefits of the P/S paradigm

- **independence** of participants
- lends itself well to distributed system development
  - de-coupled development & processing
  - (dynamic) system evolution
- **interaction** with **large number** of entities facilitated
- naturally supports **non-continuous** operations
- potential for scalability & fault-tolerance
- open for (legacy) system integration on either end

Of course it is not a **one size fits all** paradigm, but a good solution for certain kind of problems.
Other paradigms

- **request/response paradigm**
  - often also described as client/server model
  - synchronous RPC (e.g., SunRPC, CORBA, Java RMI, DCOM)
  - a blocking, point-to-point communication model
  - client must have a reference or binding to server
  - client (in p/s term: subscriber) pulls (polls) information from server (publisher)

- why are CORBA-kind of systems decreasing in popularity, especially in the web-based arena?

---

other paradigm cont.’d

- **message queuing**
  - a non-blocking invocation for point-to-point asynchronous communication
  - supports disconnected operation
  - CORBA AMI, Java Messaging, MOM efforts
  - client must have reference to server
  - P/S term: a publisher may deposite a message in a "client's" queue
other paradigm cont.'d

- **(distributed) shared memory**
  - multiple clients concurrently access (read/write - set/get) the same memory region
  - consistency enforced via locking mechanisms
  - participants are independent of one another
  - difficult to extend to interactions over wide area networks

A comparison of the above

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
<th>Arity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publish/Subscribe</td>
<td>decoupled</td>
<td>decoupled</td>
<td>n : m</td>
</tr>
<tr>
<td>Request/Response</td>
<td>coupled</td>
<td>coupled</td>
<td>1 : 1</td>
</tr>
<tr>
<td>Messaging</td>
<td>(synchronized)</td>
<td>coupled</td>
<td>1 : 1</td>
</tr>
<tr>
<td>Shared memory</td>
<td>coupled</td>
<td>decoupled</td>
<td>n : m</td>
</tr>
</tbody>
</table>
Well, but ... (some exceptions ?).

- MQ series, ...
- JavaSpaces, ...
- TIB/Rendezvous
- etc.
- industrial strength solution often claim to offer all of the above interaction models
- some may well do, however, ...
- ...others offer limited support for this & that

II. Application domains of P/S systems

Outline
- overview of P/S application domains
- requirement analysis of underlying domains
- issues and questions
Application examples & domains

- Application domains
  - alerting services
  - enterprise application integration
  - monitoring, surveillance, and control
  - network and distributed system management
  - selective information dissemination applications
  - financial applications (e.g., stock order processing)
Requirements imposed by alerting services in digital libraries

- 100 K+ of subscribers & few publishers (e.g., 10)
- low event rates (e.g., 10+ / hr)
- event size: publication record(s) (e.g., {Author, title, ...}) or entire documents
- simple subscription languages suffice (conjuncts only)
- subscriptions change infrequently (e.g., 5 / user / month)
- low notification rate (e.g., 5 / user / week)
- unreliable notification semantic sufficient
- notification propagated mostly via e-mail

Enterprise application integration

financial package (vendor X) running on mainframe

customer service package (vendor Y) running on network of PCs (UNIX & NT)

field sales personal support

EAI:
- heterogeneous platforms and information resources
- change & update propagation
- synchronization of independent copies of logically the same data
Requirements imposed by EAI

- at most 10 subscribers / publishers (correspond to systems being integrated)
- event rate = notification rate (very application dependent, e.g., request rate)
- subscription language ECA (event condition action) rules
- event sizes: size of request arguments
- subscriptions change if a new (sub-)system is added
- reliable notification semantic required

Network and distributed system management

- "When a link-down event is received, if the next link-up event for the same link is not received within 2 minutes and an alert message has not been generated in the past 5 minutes, then alert the network administrator."
Requirements imposed by NW mgmt.

- 10+ subscribers, 10,000+ publishers
- very high event rates (10,000s / sec)
- mostly small event sizes (one KB at most)
- subscription changes are infrequent (e.g., new admin.)
- complex subscription languages (causality, (real) time)
- notification rate up to 1000+ / sec (event storm)
- reliable notification semantic required
- notification propagation via NW mgmt protocols (e.g., SNMP, CMIP)
- real time constraints may be required

Selective information dissemination

- millions of subscribers and thousands of publishers
- potentially high event rates (100+ / sec)
- variable event sizes (web page(s) - attr-val. pairs)
- subscriptions are volatile, many updates and changes (e.g., 100+ / min)
- simple subscription languages suffice
- notification rate (e.g., 50 - 100 / sec)
- reliable & unreliable notification semantic
- notifications propagated via e-mail, wap, TCP, RMI, UDP, http
Summary of requirements imposed on P/S system

<table>
<thead>
<tr>
<th>Subsystem</th>
<th># sub.</th>
<th># pub.</th>
<th>evt. rate</th>
<th>evt. size</th>
<th>sub. updates</th>
<th>notif. rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alerting</td>
<td>100 K+</td>
<td>10 - 20</td>
<td>10 / hr</td>
<td>½ K - 2 MB</td>
<td>5 / user / month</td>
<td>5 / user / month</td>
</tr>
<tr>
<td>SDI</td>
<td>millions</td>
<td>1000+</td>
<td>100+ / sec. variable</td>
<td>3 / user / week</td>
<td>very volatile</td>
<td>100+ / sec.</td>
</tr>
<tr>
<td>NW mgmt.</td>
<td>10+</td>
<td>10,000+</td>
<td>10,000+ / ½ K</td>
<td>rare</td>
<td>1000+ / sec.</td>
<td></td>
</tr>
<tr>
<td>EAI</td>
<td>10</td>
<td>10</td>
<td>request rate</td>
<td>request size</td>
<td>rare</td>
<td>request rate</td>
</tr>
</tbody>
</table>

Question?

How are these requirements addressed by existing P/S systems?

How can P/S systems be characterized?

(c) 2002 Hans-Arne Jacobsen (jacobson@eceg.toronto.edu) MIE491f
Section III: P/S systems

Outline
P/S system characteristics
publication language
subscription languages
architectures and prototypes

P/S system design space

(event occurrence and notification semantic)
(subscription model / publication model)
(security)
(system model)
(inter-mediation)
(QoS)
(event delivery semantic)
(event propagation semantic)
Event occurrence and notification semantic

• An event is the occurrence of a state transition
  - sensor reaching a critical limit
  - mobile user entering a network cell
  - stock value increase / decrease / change
  - correlation between multiple value increases ...
• events happen a priori unpredictably (e.g., raising an exception, sensor limit), but
• delivery mechanisms are implemented via:
  - (remote) method invocations (e.g., registered callbacks)
  - exceptions, one-way invocations, asynchronous invocations
  - (periodic) client polls

Event delivery semantic

• push model
  - publisher initiated transmission
  - e.g., invokes operation on channel or subscriber interface
• pull model
  - a.k.a. polling
  - subscriber initiates transmission
  - e.g., requests a web-sites (dynamically updated on server)
• event can be inter-mediated through a channel (federation of), which then takes on publisher and subscriber roles
• various combinations are possible
Event propagation semantic

- **unicast** (i.e., point-to-point)
  - e-mail, pagers, cell phones
  - WAP (explicit server push in protocol)
  - TCP, UDP
  - Java RMI, IIOP
- **broadcast**
  - scoped (all receivers in certain range only)
  - un-scoped
- **multicast**
  - unreliable
  - reliable

Other characteristics

- **security support** (e.g., for subscriber registration & notification delivery, and for publisher registration, ...)
- **inter-mediation support** (one or several event channels & federations thereof)
- **application-level quality of service support**
  - expiration times (subscription leases, event time outs)
  - priorities of events
  - communication & channel reliability (best-effort, at-least-once, at-most-once, exactly-once)
  - persistence (of events)
- **logging facilities** (e.g., to capture event histories)
Classical Pub/Sub Data Models

- Channel-based
- Topic-based
- Content-based
- Type-based

Channel-based Pub/Sub Systems
**Topic-Based Model**

![Diagram of Topic-Based Model]

**Content-Based Model**

- Allows subscriptions to filter events according to their contents
- Example: subscription = (stock=IBM and price<65)
- The filter $F$ of a subscription $s$ is a stateless Boolean function that is applied to the content of an event $e$.
- An event matches a subscription $s$ if $F(e)$ is evaluated to true.
- The problem is to find all subscriptions that have $F(e)$ evaluated to true.
- Examples: ToPSS, Gryphon, SIENA, etc.
Type-Based Model

- Topics are considered as programming lang. types
- Sub-topics result from a type hierarchy
- Publications
  - Definition
    - class News (...)
    - class OntarioNews extends News (...)
  - Send an instance of a message definition
    - News m = new OntarioNews(...);

Type-Based Model

- Subscription
  - Specify the type of the message
  - e.g. "message is of type News"
  - Class msgType =
    Class.forName( News );
- Evaluation (matching)
  - if ( msgType.isInstance(m) ) { // matched!! }
- Allows "multiple specialization" (via multiple inheritance)
Subscription and publication languages

Outline
publication models & languages
  example LDAP-like publication model
  and associated subscription language
subscription model & languages
XML as subscription and publication language

Publication models & languages

- imposes a **type system** of some kind, based on
  - topics (subjects / categories) / hierarchies
  - programming language type model or DB schema
  - semi-structured data model
- publication / event as **instance** of this system
  - tagged collection of data items
  - object / data structure or DB record
  - set of attribute value pairs
Example: LDAP-like publication model (semi-structured)

- a publication described via a set of topics; a topic described via a set of attributes
- set of attributes:
  - an attribute has an unique name
  - its domain is: numeric, string, enumerated, or hierarchical
    Numeric: price  string: email
    Hierarchical: auction_category
  - domain is single or multi-valued.
    Single: price, auction_category
    Multi-valued: email, description by keywords

Example cont.'d

- set of topics:
  - a topic has an unique name
  - it is associated with a set of attributes
  - attributes in a topic may be mandatory or optional

Topic generic_auction:
  Mandatory: price, auction_category, description,
  Optional: seller_name,
Example cont.'d (a publication / an event instance)

- a set of topics
- a set of (attribute, value) pairs including
  - all mandatory attributes for the topics
  - (possibly) optional attributes for the topics.

Topics: generic_auction, antiques,
Set of (attribute, values): (price, 9000),
(auction_category, table),
(description, {“oak”, “dinning table”}),
(period, 1800), (style, Louis XV).

Example cont.'d (subscription language)

- a subscription is a conjunction of predicates.
- predicate: <attribute, operator, value>
  - =, >, <
  - contains for multi-valued attributes
  - is-kind-of for hierarchical domains

Subscription: (topic contains generic_auction) AND
(topic contains Antiques) AND
(price < 10000) AND
(style = Louis XV)
Subscription model

- **subject-based** (a.k.a. topic-based)
  - list of topics
  - hierarchy of topics
- **content-based**
  - specify subscription in a constraint-based manner
  - evaluate this expression against event content
- **typed subscription model**
  - publications are programming language typed entities
  - subscription in terms of these types and language model
  - close to content-based model

Subscription languages

- specify subscriptions to events of interest
- a.k.a. filter specification
- distinguish between
  - user defined subscriptions and
  - user profiles, (possibly) automatically generated, and maintained by system
- similar to DB query languages & trigger definitions
- some type model underlying event generation process – sub. are operators on this type model
- placement of filter (P/S system / channel & subscriber)
Subscription language: examples

- topic-based subscription
  news.toronto.rom.exhibits
- content-based subscription
  (type = news) & (location = toronto) & (object = rom) & (event = exhibit) & (subject = egypte) & (date > 3/2001) & (price < $10)
- identify event patterns in single event stream
- correlate events in multiple streams
- validity of subscription (i.e., a lease)

Further subscription language design options

- notions of time (points, intervals)
- operators on time (before, after, ...)
- changes (% increase, decrease, )
- location of objects
- movement of objects
- introduce fuzziness ("close", "about", ...)
  {2 BR, close VI arr., price about 6000 FF}

expressiveness of sub. language vs. processing complexity of matching (cf. part III)
Architectures and prototypes

Outline
• standards for P/S and event notification
• research prototypes
• industrial strength systems

Open Specifications & Standards

• CORBA Event and Notification Service
• CORBA Real-Time Notification Service
• CORBA Data-Centric Publish/Subscribe Service
• JMS – Java Messaging Service
• Java Distributed Event Specification
• JavaSpaces and Infospaces
• DCE XEMS
• MMS (Manufacturing Messaging Spec.)
• BacNet (building management & control)
CORBA Event service

- de-coupled communication between objects
- event supplies and consumer may be anonymous
- mediated via an event channel (is a CORBA object)
- supports multiple suppliers & multiple consumers
- supports push and pull interactions
- events are not object (ObV was conceived later)
- however, event data may be "any" IDL type
- no QoS, very, very limited filtering (not to say none)

module CosEventComm{
  exception Disconnected { ;
  interface PushConsumer{
    void push(in any data) raises (Disconnected);
    void disconnect_push_consumer(); }
  interface PushSupplier{
    void disconnect_push_supplier(); }
  interface PullSupplier{
    any pull() raises(Disconnected);
    any try_pull (out bool has_event) raises(...);
    void disconnect_pull_supplier();
  interface PullConsumer{
    void disconnect_pull_consumer();
  }
}
CORBA Event service cont.’d

- direct communication

- mediated communication via channel(s)

CORBA Notification Service

- extension of CORBA Event service
- adds
  - QoS features for channel
    - delivery guarantees
    - timeouts
    - priorities
  - various filtering capabilities
    - IDL-based filtering
    - constraint language based
    - manually coded filters
  - structured events (for efficient filtering)
- transaction support
Our *Push-Mi, Pull-Yu* Project

- overall **middleware platform framework** we are building to address these challenges
- support **information dissemination applications** in networked environments (mobile wireless, Internet, Intranets)
- support **network management and monitoring** applications
- framework components:
  - **TopSS**: Toronto Publish/Subscribe System
  - **X-TopSS**: XML/Xpath-based TOPSS
  - **A-TopSS**: Approximate matching based TOPSS
  - **pervasive notification engine** (unified messaging)
  - **context-aware extension** to pervasive notification engine
ToPSS architecture

Our principal research objective

What is this kernel?  How can we prove scalability?

How are higher layers build on top of kernel?
ToPSS Demo Architecture

ToPSS Implementation
A-ToPSS Web Server

Overview of industrial strength systems

- various impl. of CORBA event and notification services
- various impl. of JMS specification
- Keryx [Keryxsoft, HP]
- TIB/Rendezvous
- SonicMQ
- ...

- NetCaster & WebCaster
- BackWeb
- Marimba
- PointCast
- ...

(c) 2002 Hans-Arne Jacobsen (jacobsen@ee.c.g.toronto.edu)  MIE491f
WebCaster - Microsoft’s Web push technology

- WebCaster conceived in 1998
- (NetCaster Netscape’s pendant)
- user expresses “subscriptions” in CDF
- push is realized as smart pull (:-)) by client browser based on CDF spec.
- i.e., for the user it looks like push, however, it really is a scheduled pull (by minute, hour, day ...)
- an early application of XML !!

PointCast Inc.

- “First Internet push technology.”
- delivers profile-based customized newspaper to desktop on the Internet
- realized in a screen saver

(c) 2002 Hans-Arne Jacobsen (jacobsen@eecg.toronto.edu) MIE491f
Matching Algorithms for Internet-scale event filtering

The general matching problem

\[ e = (\text{category, book}, \text{title, Hamlet}, \text{price, $15}) \]

A fundamental principle:
Brokers network architecture is transparent to end nodes.
Two key problems to solve...

- Local matching problem
  - Efficiently match an event on a single broker

- Broker network design
  - Efficiently propagate events through a network of brokers

- Look at this in the content-based model

Challenges for local matching problem

- Large number of subscriptions
- High event rates
- High subscription update rates
- Expressiveness of the subscription language
- Size of Publication schema
- Light-weight brokers (main memory!)
Challenges for broker network design

Same as local matching problem + ...

- Geographical distribution of publishers and subscribers
- Low speed WAN
- Large number of publishers and subscribers

---

Event Matching

- An event is a set of (attribute, value) pairs
  - \{ (category, book), (title, Hamlet), (price, $20), (author, Sheakespeare) \}
- A subscription is a conjunction of predicates
  - (category = book) & (title = Hamlet)
- An event e satisfies a subscription s if every predicate in s is matched by some pair in e.
- Local event matching problem

"Given an event e and a set of subscriptions S find all subscriptions that are satisfied by e"
Problem specificity: flexibility of publication & subscription

- No pre-defined schema for events
  - (category, book), (title, Hamlet), (price, $20), (author, Shakespeare)
  - (category, antiques), (price, $80), (period, XIX)
- Multiples values for the same attribute
  - (category, [book, antiques]), (title, Hamlet), (author, Shakespeare), (price, $100)
    (period, XIX)
- Subscriptions apply on a subset of attributes
  - notify me of Hamlet sold for less than $20. (title = Hamlet) & (price < $20)
  - notify me of antique Hamlet editions
    (category = book) & (category = antiques) & (title = Hamlet)

Changes of Events and subscriptions patterns

- Changes of subscriptions patterns
  - Subscriber subjects of interest are changing over time leading to subscription skew
    - weather forecast few days before vacations
  - Events are occurring on the same subject leading to event skew
    - environmental incident detection
    - network management
  - Subscription and event skew are often timely correlated
    - US president elections
Problem definition: Summary

- Subscriptions are data, events are queries
- similar to the problem of evaluating range queries on high dimensional data space
  - events are points queries on hyper-cubes
  - events are range queries on data points
- specificity
  - subscriptions and events are incomplete
  - attributes may be multi-valued
  - both subscriptions and events are changing

Matching algorithms

- tree-based algorithms
- graph-based algorithms
- two-staged algorithms
  - predicate matching
  - subscription matching

- in this tutorial we focus on the latter kind
Predicate Matching

The Predicate matching problem

• Given a set \( P \) of predicates and an event \( e \), identify all predicates \( p \) of \( P \) which evaluate to true under \( e \).

• Example:

\[
e = \{..., (\text{price}, 5), (\text{color}, \text{white}) ...\}
\]

\( p1: \text{price} > 5; \quad p2: \text{color} = \text{red}; \quad p3: \text{price} < 4 \)

\( p1 \text{ is false} \quad p2 \text{ is false} \quad p3 \text{ is true} \)

Predicate bit vector:

\[
\begin{array}{ccc}
0 & 0 & 1 & \ldots \\
p1 & p2 & p3
\end{array}
\]

... predicate IDs
Predicate matching: top level data structure

- hash table on attribute name

\[ e = \{\ldots, (\text{price}, 5), \ldots\} \]

\[
\begin{array}{|c|c|}
\hline
\text{price} & \ldots \\
\text{color} & \ldots \\
\hline
\end{array}
\]

Predicate index: general purpose data structure

price

\[
\begin{array}{|c|c|}
\hline
\text{price} & \ldots \\
\hline
1 \rightarrow 17 & 5 \rightarrow 4 \\
0 \leftarrow 11 & 4 \rightarrow 3 \\
2 \rightarrow 6 & 5 \rightarrow 1 \\
1 \leftarrow 7 & 5 \rightarrow 9 \\
\hline
7 \rightarrow 13 & \\
\hline
\end{array}
\]

- one ordered linked list per operator
- insert, delete, match are \(O(n)\)-operations (per attribute name in \(e\) and per operator)
- alternatively, use a B-tree or B+-tree etc.
Observations about predicate value domain types

- countable domain types with small cardinality
  - integer intervals
  - collections (enums)
  - a set of tags
- Examples
  - price: [0, 1000], models variety of prices
  - color, city, state, country, size, weight
  - all tags defined in a given DTD
  - predicate domain is often context dependant, but limited in size
    - prices of cars vs. prices of groceries

Predicate matching for finite domains

price : [0, 1000] e = {..., (price, 5), ...}

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>6</th>
<th>...</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>p17</td>
<td>p4</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;</td>
<td>p11</td>
<td>p3</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>p6</td>
<td>p1</td>
<td>p13</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td>p7</td>
<td>p9</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

p1: price > 5;  p3: price < 4;  p4: price = 5;  p9: price != 5...
Predicate matching symmetries

price : [0, 1000]  e = {..., (price, 5), ...}  

=  
<  
>  
!=

Analysis

• space cost:
  \(|\text{upper} - \text{lower}| * \#\text{operators}\)

• matching cost: search cost + retrieval cost
  - no search cost, simply \(O(1)\) table look-up cost
  - retrieve all matching and non-matching predicates in
  \(|\text{upper} - \text{lower}| * \#\text{operators}\)
  - cache sensitive design: matching predicates can be retrieved by traversing the table to \textit{left} and \textit{right}
  - advantageous for densely packed tables

• insertion & deletion cost is \(O(1)\)

• far large domains and few predicates per domain, space and time penalty
Experiments and evaluation

Predicate Matching Performance
(list-based scheme)

domain sizes: 250, 10,000, 100,000

Matching Time (ms)

500 K 4.5 M

(c) 2002 Hans-Arne Jacobson (jacobson@eceg.utoronto.ca) MIE491f
Predicate Matching Performance
(table-based scheme)

domain sizes: 250, 10,000, 100,000

Predicate Matching Performance
(Tables-based vs. list-based scheme)

for the mixed domain
Predicate Matching Data Structure Size
(table-based vs. list-based scheme)

Related Work on predicate matching

- **Predicate matching**: E. Hanson extension of skip-lists, i.e., interval-skip lists - interpretation of predicate as a range query (e.g., 3 < price < 100)
Subscription Matching

Multiple one-dimension indexes

- One-dimension indexes.
  - hash tables
  - B-trees
  - Interval Skip Lists[Hanson94]
- Counting algorithm[Pereira00,SIFT00]
- Hanson algorithm[Hanson94]
- Propagation algorithm [Pereira00]
Counting algorithm

- Subscriptions consist of a set of predicates
  - S1: (2 < A < 4) & (B = 6) & (C > 4) \(\Rightarrow\) \(p_A : (2 < A < 4), p_B (B = 6), p_C (C > 4)\)
  - S1: (2 < A < 4) & (C = 3) \(\Rightarrow\) \(p_A : (2 < A < 4), p_C (C = 3)\)
- A Subscription matches the event if all its predicates are satisfied.
- Idea: Count the number of satisfied predicates per subscription

Data structures for the counting algorithm

<table>
<thead>
<tr>
<th>TOTAL NUMBER</th>
<th>COUNT</th>
<th>Predicate vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>3</td>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
<td>2</td>
<td>S1</td>
</tr>
<tr>
<td>S3</td>
<td>2</td>
<td>S1</td>
</tr>
</tbody>
</table>
**Counting algorithm**

Indexes

- A
- B
- C

**Predicate vector**

E: (A, 5), (B, 6)

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Total number**

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Subscription Matching**

**Bit-matrix based**

- Subscription IDs
  - Pred. bit vec
    - 1
    - 0
    - 1
    - 1
    - 0
  - Hit vector
  - Preds-per-sub

**List based**

- Pred. bit vec
  - \( s_{17} \) \( \rightarrow s_4 \)
  - \( s_{11} \) \( \rightarrow s_3 \)
  - \( s_6 \) \( \rightarrow s_1 \)
  - \( s_7 \) \( \rightarrow s_9 \)
  - \( \rightarrow s_{13} \)

- Hit vector
- Preds-per-sub

+ Sub-pred association to support deletion
Counting algorithm: search cost

- Search cost is sensitive to predicate selectivity:
  - 1: Compute all satisfied predicates
    - $d \cdot \ln(P)$ (tree-based schemes)
    - $P$ is the avg number of predicates per dimension
  - 2: Counting satisfied predicates
    - $\text{Unit} \cdot N \cdot s_p \cdot p_{\text{avg match}}$
    - $N$ is the number of subscriptions
    - $s_p$ is the avg number of predicates per subscription
    - $p_{\text{avg match}}$ is a avg probability of a predicate to be satisfied

Counting algorithm: cost of data structures

- Insertion of a subscription $S$
  - updating indexes
    - $s_p \cdot \ln(P)$
  - Updating subscriptions lists
    - $s_p$
- Space cost
  - $N \cdot s_p$
Counting algorithm: Summary

- Handles incomplete subscriptions
- Schema changes are easy
- Low subscription insertion cost
- Search cost:
  - linear with the subscriptions space
  - depends on the average predicate selectivity
- Very sensitive to predicates having low filtering power