Peer-to-Peer Grid Databases for Service and Resource Discovery

Wolfgang.Hoschek@cern.ch
CERN IT/DB
EDG WP2

Outline

- Describe how to…
  - bootstrap, query and publish content
  - to a dynamic information space
  - maintained by a web of self-describing network interfaces
- Show how to support…
  - expressive general-purpose queries
    - e.g. for service discovery
    - e.g. using XQuery, SQL, XPath or custom query language
  - over a view that integrates autonomous dynamic database nodes
    - Tuple set partitioned for scalability, availability, performance, security
    - Query results reflect current state
  - from any distributed system topology
    - e.g. ring, tree, graph, hybrids

⇒ Result: Unified Peer-to-Peer Database Framework (UPDF) and Peer Database Protocol (PDP)

Introduction

- Computer systems: Tools for WAN communication and collaboration
  - Colleagues (and programs) with shared interests can work together
    - with less respect to location of people, devices and machinery
  - Departmental team complemented by cross-organizational teams
    - Virtual Organizations
  - Appears natural to science communities
    - Tradition in innovation from partnerships across admin. boundaries
- Core concepts and technologies for global infrastructures
  - Grid Computing
  - Peer-to-Peer Computing
  - Distributed Databases
  - Web Services
- Convergence promises powerful emerging synergies

Grid Computing

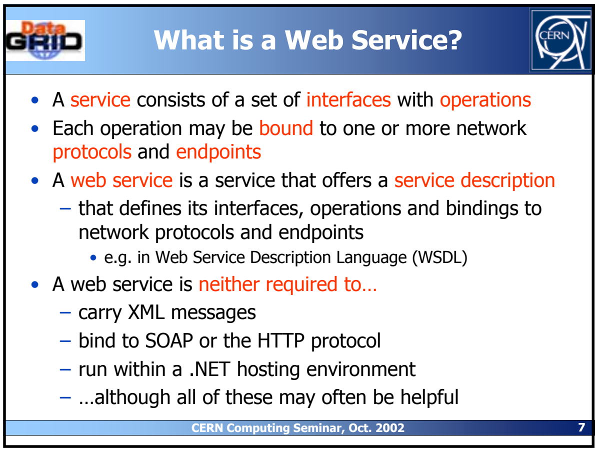
- Support flexible, secure, coordinated information sharing
  - among dynamic collections of individuals, institutions and resources
  - Also includes remote access to computers, software and devices required by collaborative problem solving
  - Join loosely coupled people and resources from multiple organizations
- Grid = Collaborative distributed Internet system with…
  - large scale, heterogeneity, lack of central control
  - multiple autonomous administrative domains
  - unreliable components and frequent dynamic change

Services

- Most functionality available as 3rd party libraries, frameworks, tools
  - Key skill of software developers
    - No more: hard core programming
    - Ability to find, assess & integrate building blocks from many 3rd parties
- Next: Components network-attached → network services
  - for use by the general public, collaborators or customers
  - Application Service Providers run and maintain reliable services on behalf of clients through hosting environments
- Client Configuration
  - Hard-wire the location, interface and other properties of remote services into the local application
  - Straightforward but inflexible
- Loosely coupled decentralized systems
  - Solutions must adapt to changing conditions
  - Enable programs to search the Internet for alternative but similar services and dynamically substitute these (find, assess & integrate)

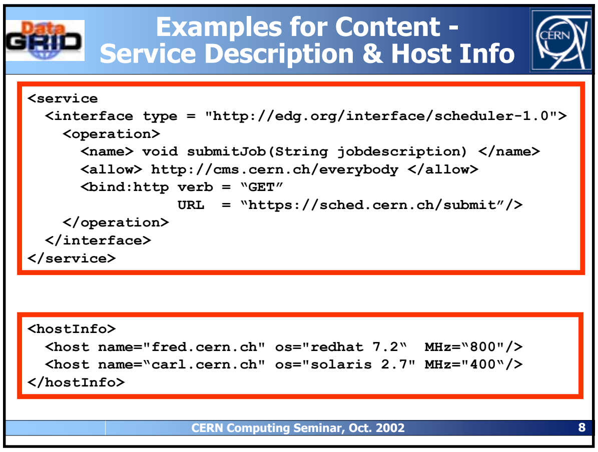
Web Service Discovery

- Programs no longer configured with static information
- Programs more flexible, adaptive and powerful
  - by querying Internet databases (registries) at runtime
  - to discover services and related metadata
- Services can publish (advertise) themselves and related metadata
  - enabling the assembly of higher-level components
- Example (European Data Grid @ CERN)
  - data-intensive High Energy Physics analysis application
  - looks for remote services
  - that exhibit a suitable combination of characteristics, including
  - service interfaces, operations and protocols,
  - network load, available disk quota, access rights, and perhaps Quality of Service (QoS) and monetary cost



What is a Web Service?

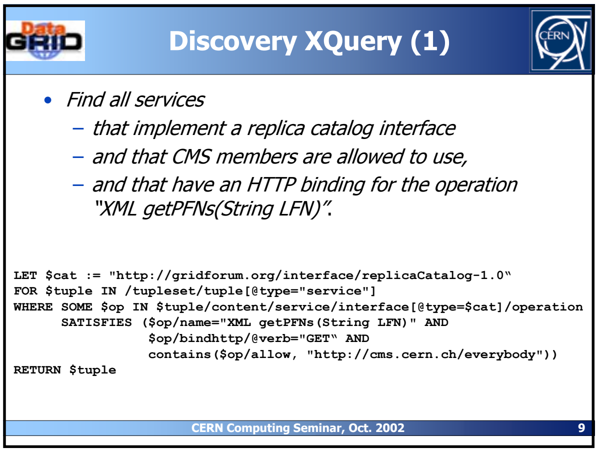
- A service consists of a set of interfaces with operations
- Each operation may be bound to one or more network protocols and endpoints
- A web service is a service that offers a service description
  - that defines its interfaces, operations and bindings to network protocols and endpoints
    - e.g. in Web Service Description Language (WSDL)
- A web service is neither required to…
  - carry XML messages
  - bind to SOAP or the HTTP protocol
  - run within a .NET hosting environment
  - …although all of these may often be helpful



Examples for Content - Service Description & Host Info

```
<service
  <interface type = "http://edg.org/interface/scheduler-1.0">
    <operation>
      <name> void submitJob(String jobdescription) </name>
      <allow> http://cms.cern.ch/everybody </allow>
      <bind:http verb = “GET”
                 URL  = “https://sched.cern.ch/submit”/>
    </operation>
  </interface>
</service>
```

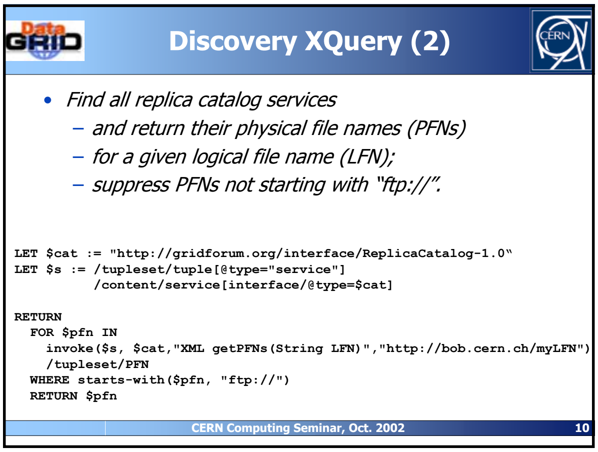
```
<hostInfo>
  <host name="fred.cern.ch" os="redhat 7.2“  MHz=“800"/>
  <host name=“carl.cern.ch" os="solaris 2.7" MHz="400“/>
</hostInfo>
```



Discovery XQuery (1)

- *Find all services*
  - *that implement a replica catalog interface*
  - *and that CMS members are allowed to use,*
  - *and that have an HTTP binding for the operation "XML getPFNs(String LFN)”.*

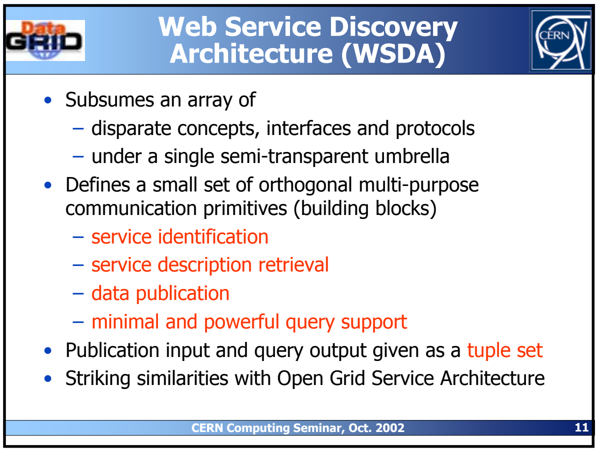
```
LET $cat := "http://gridforum.org/interface/replicaCatalog-1.0“
FOR $tuple IN /tupleset/tuple[@type="service"]
WHERE SOME $op IN $tuple/content/service/interface[@type=$cat]/operation
      SATISFIES ($op/name="XML getPFNs(String LFN)" AND
                 $op/bindhttp/@verb="GET“ AND
                 contains($op/allow, "http://cms.cern.ch/everybody"))
RETURN $tuple
```



Discovery XQuery (2)

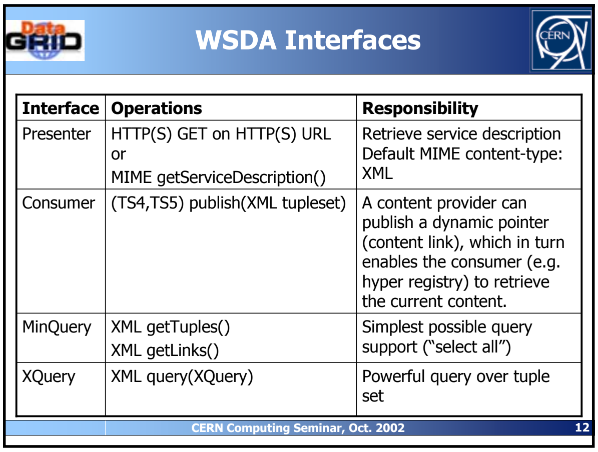
- *Find all replica catalog services*
  - *and return their physical file names (PFNs)*
  - *for a given logical file name (LFN);*
  - *suppress PFNs not starting with “ftp://”.*

```
LET $cat := "http://gridforum.org/interface/ReplicaCatalog-1.0“
LET $s := /tupleset/tuple[@type="service"]
          /content/service[interface/@type=$cat]

RETURN
  FOR $pfn IN
    invoke($s, $cat,"XML getPFNs(String LFN)","http://bob.cern.ch/myLFN")
    /tupleset/PFN
  WHERE starts-with($pfn, "ftp://")
  RETURN $pfn
```



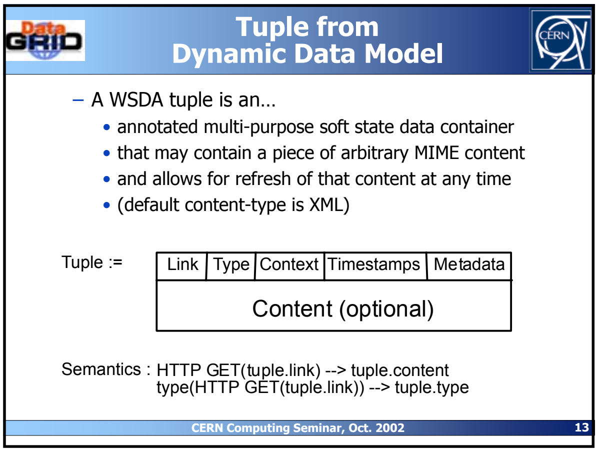
Web Service Discovery Architecture (WSDA)

- Subsumes an array of
  - disparate concepts, interfaces and protocols
  - under a single semi-transparent umbrella
- Defines a small set of orthogonal multi-purpose communication primitives (building blocks)
  - service identification
  - service description retrieval
  - data publication
  - minimal and powerful query support
- Publication input and query output given as a tuple set
- Striking similarities with Open Grid Service Architecture



WSDA Interfaces

Interface	Operations	Responsibility
Presenter	HTTP(S) GET on HTTP(S) URL or MIME getServiceDescription()	Retrieve service description Default MIME content-type: XML
Consumer	(TS4,TS5) publish(XML tupleset)	A content provider can publish a dynamic pointer (content link), which in turn enables the consumer (e.g. hyper registry) to retrieve the current content.
MinQuery	XML getTuples() XML getLinks()	Simplest possible query support (“select all”)
XQuery	XML query(XQuery)	Powerful query over tuple set



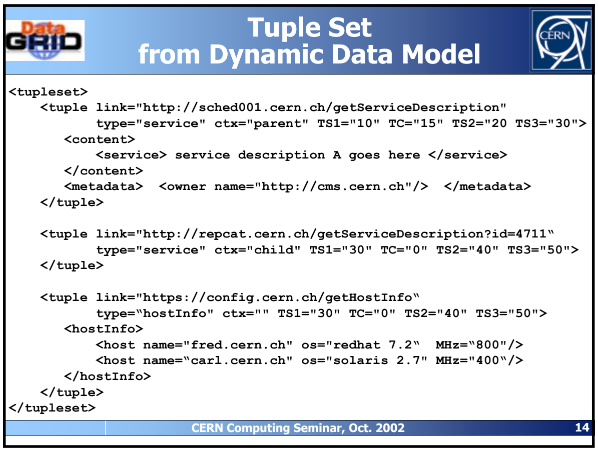
Tuple from Dynamic Data Model

- A WSDA tuple is an...
  - annotated multi-purpose soft state data container
  - that may contain a piece of arbitrary MIME content
  - and allows for refresh of that content at any time
  - (default content-type is XML)

Tuple :=

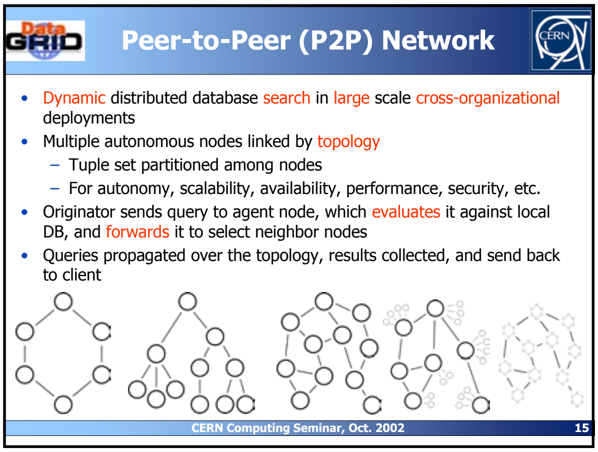
Link	Type	Context	Timestamps	Metadata
Content (optional)				

Semantics : HTTP GET(tuple.link) --> tuple.content
type(HTTP GET(tuple.link)) --> tuple.type



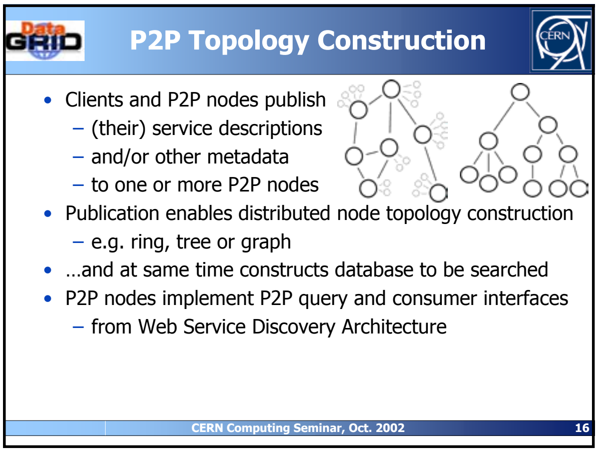
Tuple Set from Dynamic Data Model

```
<tupleset>
    <tuple link="http://sched001.cern.ch/getServiceDescription"
           type="service" ctx="parent" TS1="10" TC="15" TS2="20 TS3="30">
       <content>
           <service> service description A goes here </service>
       </content>
       <metadata>  <owner name="http://cms.cern.ch"/>  </metadata>
    </tuple>

    <tuple link="http://repcat.cern.ch/getServiceDescription?id=4711"
           type="service" ctx="child" TS1="30" TC="0" TS2="40" TS3="50">
    </tuple>

    <tuple link="https://config.cern.ch/getHostInfo"
           type="hostInfo" ctx="" TS1="30" TC="0" TS2="40" TS3="50">
       <hostInfo>
           <host name="fred.cern.ch" os="redhat 7.2"  MHz="800"/>
           <host name="carl.cern.ch" os="solaris 2.7" MHz="400"/>
       </hostInfo>
    </tuple>
</tupleset>
```



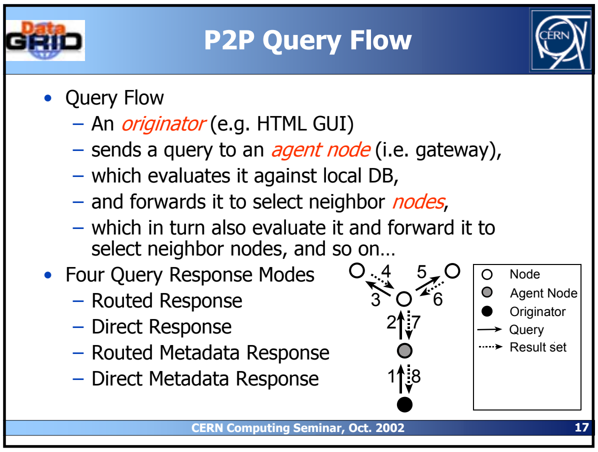
Peer-to-Peer (P2P) Network

- Dynamic distributed database search in large scale cross-organizational deployments
- Multiple autonomous nodes linked by topology
  - Tuple set partitioned among nodes
  - For autonomy, scalability, availability, performance, security, etc.
- Originator sends query to agent node, which evaluates it against local DB, and forwards it to select neighbor nodes
- Queries propagated over the topology, results collected, and send back to client



P2P Topology Construction

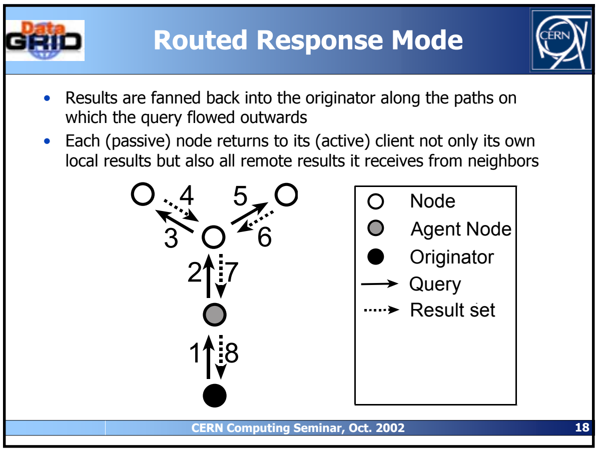
- Clients and P2P nodes publish
  - (their) service descriptions
  - and/or other metadata
  - to one or more P2P nodes
- Publication enables distributed node topology construction
  - e.g. ring, tree or graph
- ...and at same time constructs database to be searched
- P2P nodes implement P2P query and consumer interfaces
  - from Web Service Discovery Architecture



P2P Query Flow

- Query Flow
  - An *originator* (e.g. HTML GUI)
  - sends a query to an *agent node* (i.e. gateway),
  - which evaluates it against local DB,
  - and forwards it to select neighbor *nodes*,
  - which in turn also evaluate it and forward it to select neighbor nodes, and so on...
- Four Query Response Modes
  - Routed Response
  - Direct Response
  - Routed Metadata Response
  - Direct Metadata Response

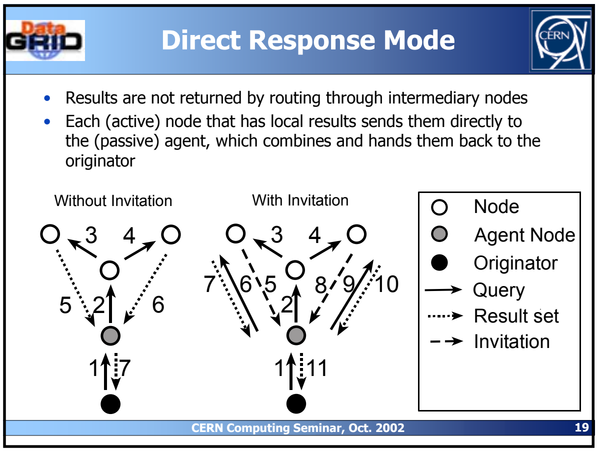
○ Node
● Agent Node
● Originator
→ Query
⋯→ Result set



Routed Response Mode

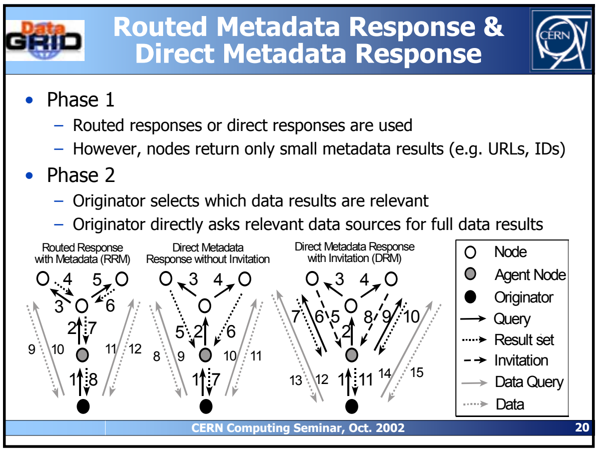
- Results are fanned back into the originator along the paths on which the query flowed outwards
- Each (passive) node returns to its (active) client not only its own local results but also all remote results it receives from neighbors

○ Node
● Agent Node
● Originator
→ Query
⋯→ Result set



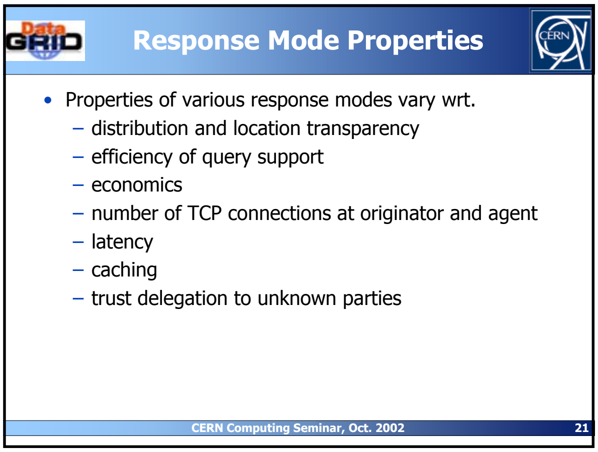
Direct Response Mode

- Results are not returned by routing through intermediary nodes
- Each (active) node that has local results sends them directly to the (passive) agent, which combines and hands them back to the originator



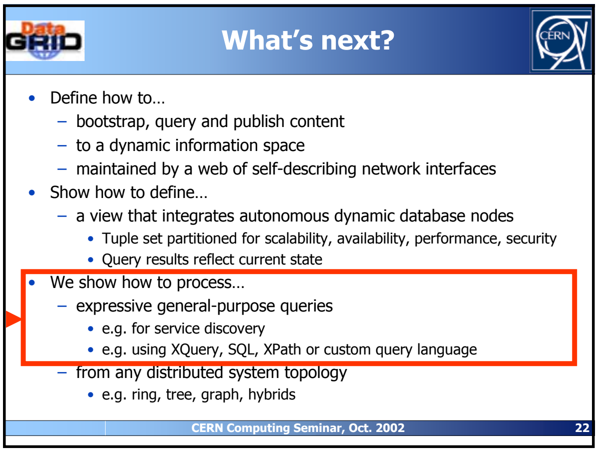
Routed Metadata Response & Direct Metadata Response

- Phase 1
  - Routed responses or direct responses are used
  - However, nodes return only small metadata results (e.g. URLs, IDs)
- Phase 2
  - Originator selects which data results are relevant
  - Originator directly asks relevant data sources for full data results



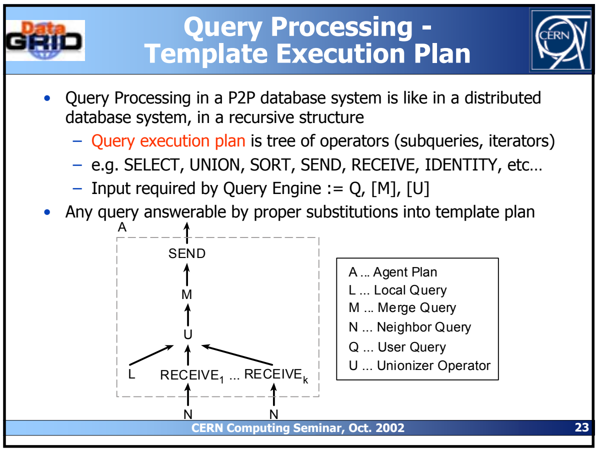
Response Mode Properties

- Properties of various response modes vary wrt.
  - distribution and location transparency
  - efficiency of query support
  - economics
  - number of TCP connections at originator and agent
  - latency
  - caching
  - trust delegation to unknown parties



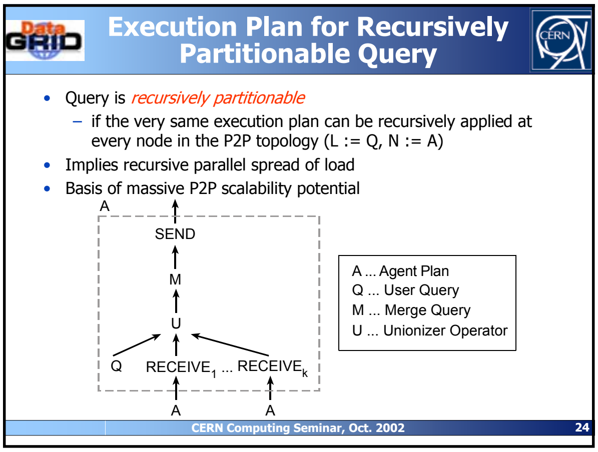
What's next?

- Define how to…
  - bootstrap, query and publish content
  - to a dynamic information space
  - maintained by a web of self-describing network interfaces
- Show how to define…
  - a view that integrates autonomous dynamic database nodes
    - Tuple set partitioned for scalability, availability, performance, security
    - Query results reflect current state
- We show how to process…
  - expressive general-purpose queries
    - e.g. for service discovery
    - e.g. using XQuery, SQL, XPath or custom query language
  - from any distributed system topology
    - e.g. ring, tree, graph, hybrids



Query Processing - Template Execution Plan

- Query Processing in a P2P database system is like in a distributed database system, in a recursive structure
  - Query execution plan is tree of operators (subqueries, iterators)
  - e.g. SELECT, UNION, SORT, SEND, RECEIVE, IDENTITY, etc…
  - Input required by Query Engine := Q, [M], [U]
- Any query answerable by proper substitutions into template plan



Execution Plan for Recursively Partitionable Query

- Query is *recursively partitionable*
  - if the very same execution plan can be recursively applied at every node in the P2P topology (L := Q, N := A)
- Implies recursive parallel spread of load
- Basis of massive P2P scalability potential

Query Types and Examples wrt. Recursive Partitioning

☺ Simple Query: Recursively partitionable with M=IDENTITY, U=UNION
- Find all (available) services.
- Find all services that implement a replica catalog service interface and that CMS members are allowed to use, and that have an HTTP bindings for the replica catalog operation "XML getPFNs(String LFN)".
- Find all CMS replica catalogs and return their physical file names (PFNs) for a given logical file name (LFN); suppress PFNs not starting with "ftp://".

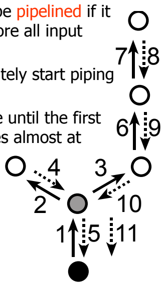
☺ Medium Query: Recursively partitionable
- Return the number of replica catalog services.

☹ Complex Query: Not recursively partitionable (e.g. some JOINs)
- Find all (execution service, storage service) pairs where both services of a pair live within the same domain
- Find all hosts that run more than one replica catalog with CMS as owner

Pipelining

- Originator often happy to work with a few early results
  - as long as they arrive quickly and reliably
- A query (an operator implementation) is said to be pipelined if it can already produce at least one result tuple before all input tuples have been seen
- For some query types the originator can immediately start piping in results (at moderate performance rate)
- For other query types it must wait for a long time until the first result becomes available (the full result set arrives almost at once, however)

Query Type	Supports Pipelining?
Simple Query	Yes
Medium Query	Maybe
Complex Query	Typically No



Timeout

- There comes a time when a user is no longer interested in query results
  - no matter whether any more results might be available
- The query roaming the network and its response traffic should fade away after some time
  - Hence: Abort timeout (deadline) attached to query
  - *"I will ignore (the rest of) your result set if I have not received it before 12:00:00 today."*
- The problem
  - Ensure that a maximum of results can be delivered reliably within the time frame desired by user

Static Loop Timeout and Dynamic Abort Timeout

- Dynamic Timeout
  - Timeout value intended to be decreased at each hop
  - Nodes further away from the originator may time out earlier than nodes closer to the originator
  - Mandatory for abort timeout of non-pipelined query
    - Provides some safety time window for the partial results of any node to flow back across multiple hops to the originator
    - Good Dynamic Scheme: Exponential decay with halving
- Static timeout
  - Timeout value remains unchanged across node hops
  - Mandatory for reliable loop detection in query routes (loop timeout)

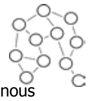
Query vs. Query Scope - Motivation

- Goal 1
  - exploit several independent information sources as if they were a single source
- Goal 2
  - In practice, sufficient (and more efficient) to consider only a subset of all tuples (service descriptions) from a subset of nodes

- Example
  - Query only searches tuples (services) within the scope of the domain "cern.ch" and ignores the rest of the world

⇒Separate (logical) query and (physical) query scope

Query and Query Scope

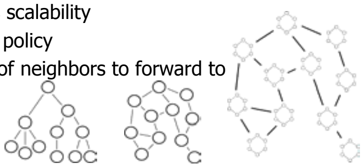


- Query
  - is formulated against a global database view
  - is insensitive to link topology and deployment model
  - To a query the set of tuples appears as a single homogenous database
  - even though the set may be (recursively) partitioned across many nodes and databases
  - → In a relational or XML environment, the set of all tuples appears as a single, very large, table or XML document, respectively
- Query scope
  - specifies the input tuple set of query
  - is used to navigate and prune the link topology and filter on attributes of the deployment model

Scope Specification by Neighbor Selection Policy

- So far implicitly assumed a broadcast model (over TCP)
  - a node forwards a query to all neighbor nodes
- Snowballing (epidemic, flooding) effect
  - Overall bandwidth consumption grows exponentially with query radius
  - produces enormous stress on the network
  - drastically limits scalability

⇒Neighbor selection policy
  - select a subset of neighbors to forward to



Neighbor Selection Query

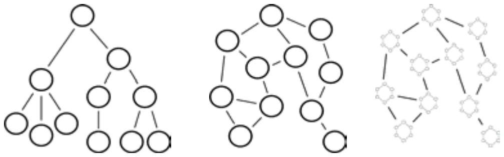
- User defines neighbor selection query (XQuery)
  - Input: tuple set of current node
    - i.e. any static or dynamic data published to node
  - Output: subset indicating select nodes for forwarding

```
RETURN /tupleset/tuple[@type="service"
  AND content/service/interface[@type="Consumer-1.0"]
  AND content/service/interface[@type="XQuery-1.0"]]
```

- Used for Smart Dynamic Routing
  - Broadcast, random selection, domain, security, access control filters, latency and bandwidth filters
  - In tree topology select child nodes only (schedulers)
  - Hierarchical Namespace (DNS, LDAP), JXTA Groups, Nodes subscribe to Query Types, Indexing, Caching,…

More Means for Scope Specification

- Abort Timeout
- Radius
  - a measure of path length
  - the maximum number of hops a query is allowed to travel on any given path
  - decreased by one at each hop



Peer Database Protocol (PDP)

- Supports all features of P2P framework
- Query and publish-subscribe functionality
- Fully based on BEEP IETF standard
  - SOAP can be carried over BEEP [Marshall:2002a]
  - BEEP can be carried over any reliable transport (TCP is merely the default)
- Transaction
  - one or more discrete message exchanges related to the same query
- Messages
  - QUERY, RECEIVE, SEND, INVITE, CLOSE

PDP Properties

- Low latency, pipelining, early and/or partial result set retrieval due to synchronous pull, and result set delivery in one or more variable sized batches
- Efficiency
  - due to asynchronous push with delivery of multiple results per batch
- Resource consumption and flow control per query
  - due to the use of a distinct channel per transaction
- Scalable
  - due to application multiplexing, which allows for very high query concurrency and very low latency, even in the presence of secure TCP connections

Permitted Message Exchanges

```
• MSG_QUERY   --> RPY_OK | ERR
• MSG_RECEIVE --> RPY_SEND |
                  (ANS_SEND [0:N], NULL) |
                  ERR
• MSG_INVITE  --> RPY_OK | ERR
• MSG_CLOSE   --> RPY_OK | ERR
```

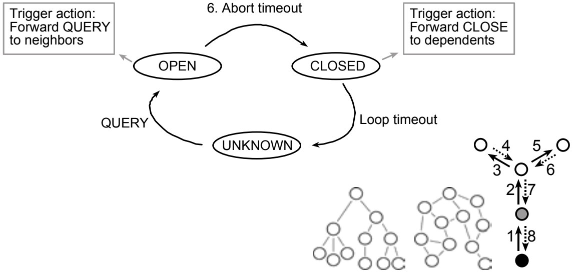
- Supports synchronous (pull) and asynchronous (push)
- Supports batched iterators
  - RECEIVE/SEND batches of at least N and at most M results from the (remainder of the) result set

Example Message Exchanges

Routed Synchr. Response	Routed Async. Response	Direct Synchr. Response	Direct Async Response
--> QUERY	--> QUERY	--> QUERY	--> QUERY
--> RECEIVE	--> RECEIVE		
<-- SEND	<-- SEND	<-- INVITE	<-- INVITE
--> RECEIVE	<-- SEND	--> RECEIVE	--> RECEIVE
<-- SEND	--> CLOSE	<-- SEND	<-- SEND
--> CLOSE		--> RECEIVE	<-- SEND
		<-- SEND	--> CLOSE
		--> CLOSE	

Node State Transitions

1. CLOSE received
2. SEND exhausts result set
3. INVITE not accepted (Direct Response non empty resultset)
4. True (Direct Response empty local result set)
5. Various errors
6. Abort timeout


Trigger action: Forward QUERY to neighbors
Trigger action: Forward CLOSE to dependents
OPEN
CLOSED
QUERY
UNKNOWN
Loop timeout
4
5
3
6
2
7
1
8

Example Query Message

```
<MSG_QUERY transactionID = "12345">
  <query>
    <userquery> RETURN /tupleset/tuple </userquery>
    <mergequery unionizer="UNION"> RETURN /tupleset/tuple
    </mergequery>
  </query>

  <scope loopTimeout="2000" abortTimeout="1000‟
         logicalRadius = "7" physicalRadius = "4‟
         maxResults = "100"  maxResultsBytes = "100000">
         <neighborSelectionQuery>        <!-- broadcast -->
            RETURN /tupleset/tuple[@type="service‟
              AND content/service/interface[@type="Consumer"]
              AND content/service/interface[@type="XQuery"]]
         </neighborSelectionQuery>
  </scope>
  <options> <responseMode> routed </responseMode> </options>
</MSG_QUERY>
```

Example Receive & Send Message

```
<MSG_RECEIVE transactionID = "12345">
  <mode minResults="1" maxResults="10"> synchronous </mode>
</MSG_RECEIVE>

<RPY_SEND transactionID = "12345">
  <data nonBlockingAvailable = "-1" estimatedAailable = "-1">
    <tupleset>
      <tuple link="http://sched.infn.it/getServiceDesc‟
             type="service" ctx="child" TS1="20" TC="25‟
             TS2="30" TS3="40">
        <content>
          <service> service description goes here </service>
        </content>
      </tuple>
      ... more tuples can go here ...
    </tupleset>
  </data>
</RPY_SEND>
```

Example Answer, Invite, Close, OK and Error Message

```
<ANS_SEND transactionID = "12345">
  structure is identical to RPY_SEND (see above) ...
</ANS_SEND>

<MSG_INVITE transactionID = "12345">
  <avail nonBlockingAvailable="50‟ estimatedAvailable="100"/>
</MSG_INVITE>

<MSG_CLOSE transactionID = "12345" code="555">
  maximum idle time exceeded
</MSG_CLOSE>

<RPY_OK transactionID = "12345"/>

<ERR transactionID = "12345" code="550">
  transaction identifier unknown
</ERR>
```

Related Work (1)

- Gnutella, Freenet, Tapestry, Chord, Globe, other P2P systems
  - Assume *simple* query (e.g. lookup by unique ID)
  - Mostly clever distributed hash table lookup
- DNS, LDAP, MDS
  - Assume *simple* query from hierarchical namespace
  - Do not support graph topology, flexible neighbor selection, dynamic timeout, radius, loop detection
- Distributed Database Systems
  - Assume homogeneous administration, security, network, etc.
  - Do not support autonomous nodes (read: Internet, Grid)
  - Grid: heterogeneity, scale, multiple autonomous administrative domains, unreliable components and frequent dynamic change
- We support all query types, including expressive general-purpose query languages (e.g. XQuery, SQL, XPath)

Related Work (2)

- UDDI (Universal Description, Discovery and Integration)
  - an emerging industry standard that defines a business oriented access mechanism to a centralized registry holding XML based WSDL service descriptions
  - No dynamic data model & soft state. Rudimentary query support. Not scalable (no P2P network)
- JXTA
  - Defines six stateless best-effort protocols for ad hoc, pervasive, and multi-hop P2P computing
  - Simple queries that are unreliable, stateless, non-pipelined, and non-transactional (limits scalability, efficiency and applicability)
  - Lacking expressive means for query scoping, neighbor selection and timeouts, it is unclear how chained rendezvous peers can form a search network
  - We believe that JXTA Peer Groups, JXTA search and publish/subscribe can be expressed within our UPDF framework, but not vice versa

Related Work (3)

- Grid Monitoring Architecture (GMA)
  - Efficient monitoring of distributed components,
    - for example to allow for fault detection and performance prediction
  - Briefly sketches three interactions for transferring data between producers and consumers
    - publish/subscribe, query/response and notification
  - No query language, no data model, no network protocol
  - No multi-hop (P2P) network
    - hence no loop detection, scoping, timeouts and neighbor selection
- Open Grid Services Architecture (OGSA)
  - Striking similarities with WSDA, in spirit and partly also in design
  - Not scalable (no P2P network)
- Our uniformity and wide applicability is distinguished from related work, which
  - addresses some but not all problems
  - and does not propose a unified framework

Summary

- Query and publish content to a dynamic information space
  - maintained by web of self-describing network interfaces
- Q: Can we devise a framework …
  - for general-purpose query support
  - in large heterogeneous distributed systems
  - spanning many administrative domains?
- Q: Can we devise a a unified P2P database framework that allows to express specific discovery applications for a wide range of …
  - data types (typed or untyped XML, any MIME type)
  - node topologies (e.g. ring, tree, graph, hybrids)
  - query languages (e.g. XQuery, SQL, LDAP, XPath)
  - query response modes (e.g. Routed, Direct and Metadata Responses)
  - neighbor selection policies (e.g. in the form of an XQuery)
  - pipelining characteristics, timeout and other scope options?
- → Unified P2P DB Framework (UPDF) and Peer Database Protocol (PDP)

Questions?

- More information
  - http://cern.ch/grid-data-management/
  - http://www.edg.org
- Contacts
  - wolfgang.hoschek@cern.ch
- Thanks for your attention!