DATABASE APPLICATIONS ARCHETECTURE

David Dykstra, Igor Mandrichenko

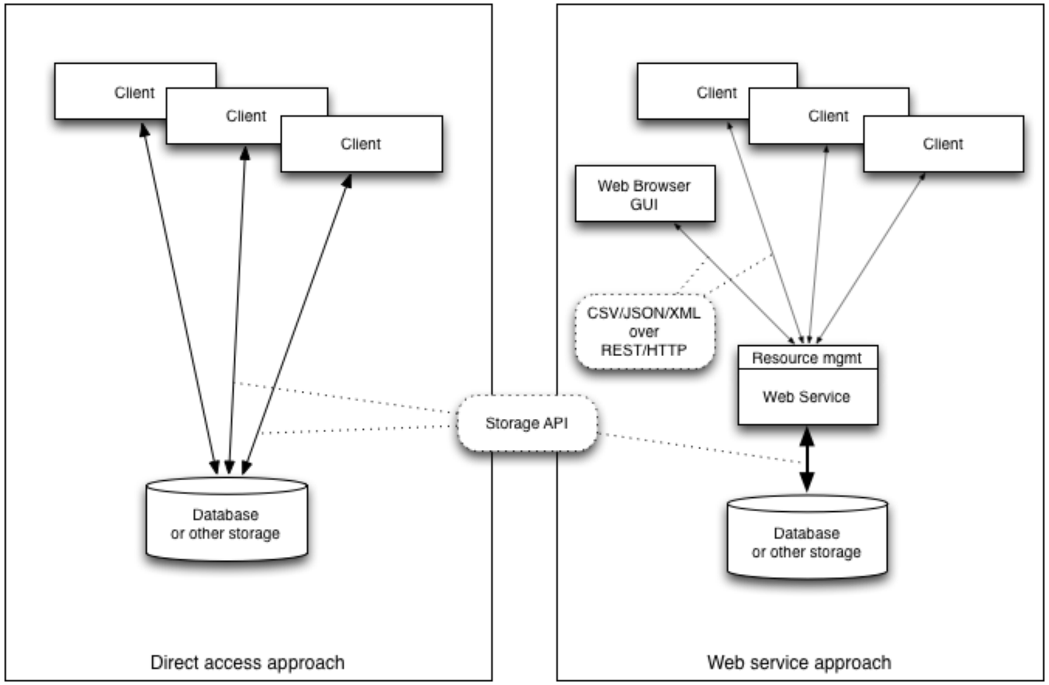
STA meeting, June 25, 2014

Goals

- Support computing needs of experiments
- Develop short to medium term strategy – 3-5 years
- Trends
  - Database are becoming more popular
  - Data intensity
  - Grid computing
    - Massively parallel access
    - Remote access
  - Internet technologies
- Robust, reliable, scalable, manageable architecture for data access

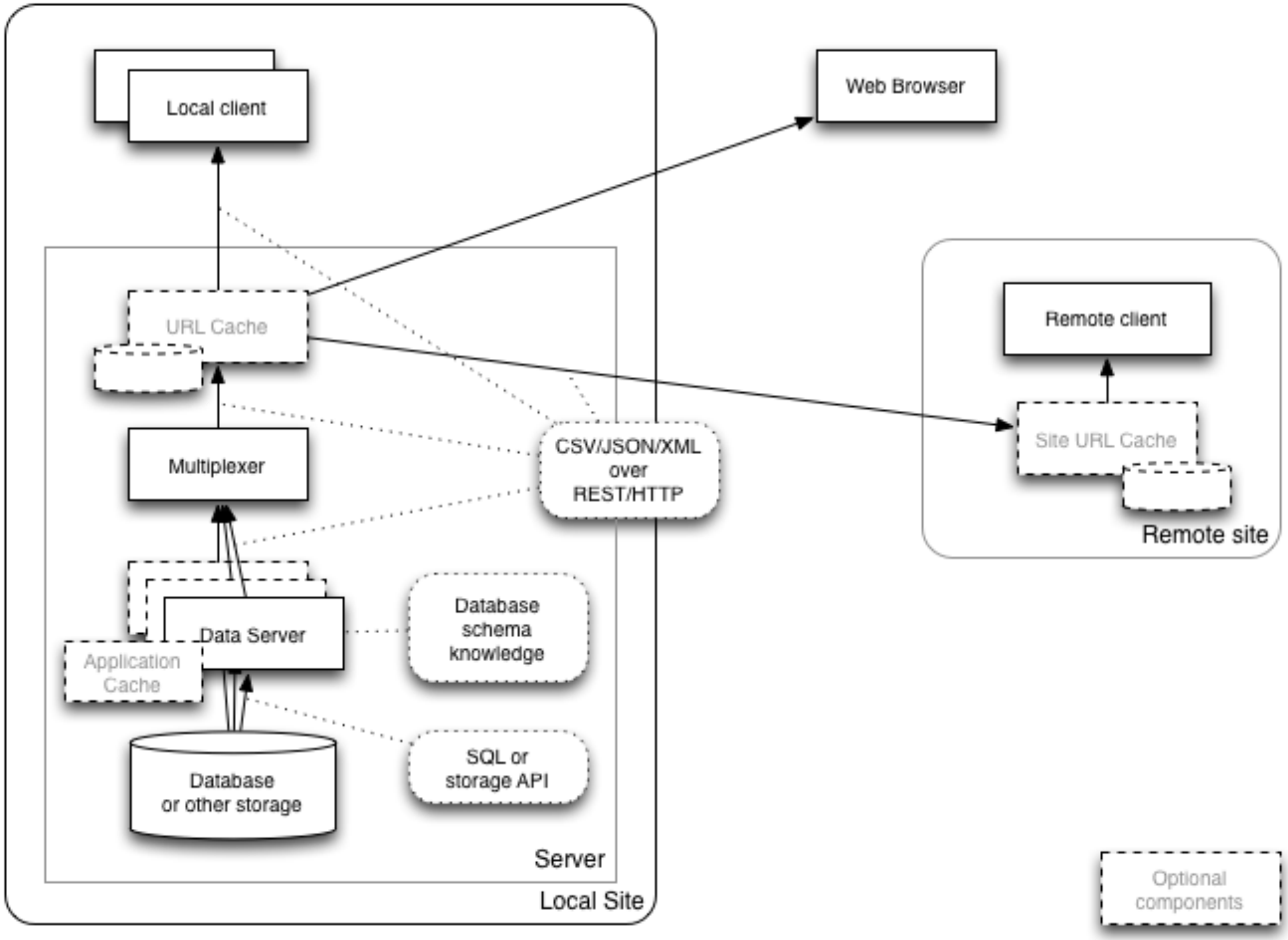
Web services, not direct access

- Fundamental idea:
  - Do not allow direct access to the storage (database), put a web service between the client and the storage



- Reuse technologies developed for the Internet
  - Protocols, interfaces, libraries, tools, frameworks, knowledge
- Decouple the client and the storage implementation
  - Hide the details, complexity and intensity of the storage communication behind the server
  - Independent implementation of the client and the server
- Add resource management layer

Current Architecture



Major features

- Use of common Internet standards (W3C, IETF)
  - HTTP, HTTPS, CSV, JSON, XML
- Common web applications development frameworks, tools
  - WSGI, Apache httpd, squid, etc.
- Redundant web services infrastructure
  - Performance, availability, flexibility, resource management
- Modular design – optional components can be plugged in or removed

Frontier

- Started around 2004 for D0 as a database web application framework
  - Same idea: channel database communication through HTTP
    - No SQL communication, DB schema is hidden
  - Emphasis on URL caching
  - Currently is used as CDF Frontier
- Redesigned for CMS
  - Send SQL over HTTP, expose schema to the client
- Continued development
  - Option to hide schema from the client
  - Focus on site caching, client side multiplexing, robustness on large scale

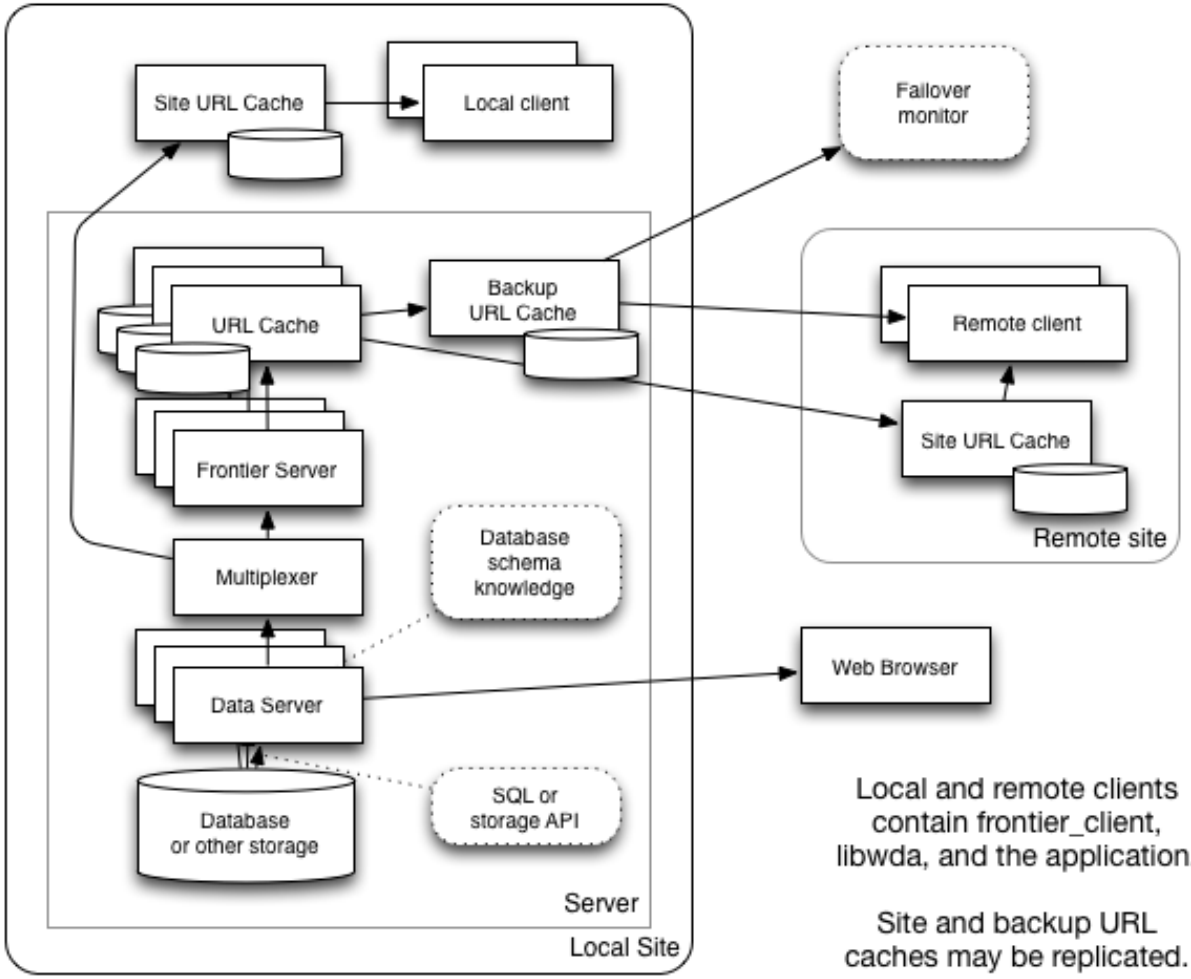
Major features

- Client side multiplexing
  - Client chooses from multiple locations (caches) to request data from
    - Round-robin
    - Primary/backup
    - Multiple server sites
    - WLCG site cache discovery standard and implementation – work in progress
- Additional layer of protocol on top of HTTP standard
  - Frontier client can talk only to Frontier server
- Failover monitoring
- SQL over HTTP
- All of these features can be added to current architecture without Frontier

Proposal

- If direct SQL access is required, use Frontier
- Add Frontier on top of the architecture when:
  - Existing architecture reaches its scalability limits
  - URL caching is possible and beneficial
  - Caching infrastructure requires client side multiplexing

How to add Frontier



Local and remote clients contain frontier_client, libwda, and the application.

Site and backup URL caches may be replicated.

End

- Remaining slides could be useful for the discussion but are not part of the presentation.

Redundant Web Services Infrastructure

- Multiple redundant application and data servers, running on real and virtual machines
  - Performance
  - Availability
- Access multiplexing
  - HTTP redirector for interactive applications
  - HTTP proxy for data applications
- Used to run about dozen different applications, data and interactive

On caching

Request is the fundamental resource. Minimize the number of requests coming through the system and hitting the database.

- Caching: re-use the data, retrieved or computed previously
  - Can significantly improve system performance
  - Or can decrease the performance and increase the load on the resources if the data is not cacheable

- When caching is good:
  - Data must be a deterministic function of the request and the request time
  - Dependency on the request time must be slow
    - Cache preemption
    - Data do not change between subsequent requests
    - It is easier to save data than to re-retrieve or re-compute

- Examples:
  - State of the detector for run N – cacheable
  - Current state of the detector – not cacheable

How to cache

- Client side
  - Do not ask for same data twice
  - Ask more than you immediately need and use it later
  - Application dependent

- URL caching
  - If URL is good key for data, use Internet technologies (caching proxy) to cache data
  - Application independent

- Server side caching
  - Cache intermediate data and produce output from it
    - Data = F(R) – not cacheable
    - F = F(G(R)) but G(R) is cacheable
    - Application dependent