FTS reliability

Paolo Tedesco

WLCG Service Reliability Workshop, CERN

November 2007

www.eu-egee.org

Information Society and Media

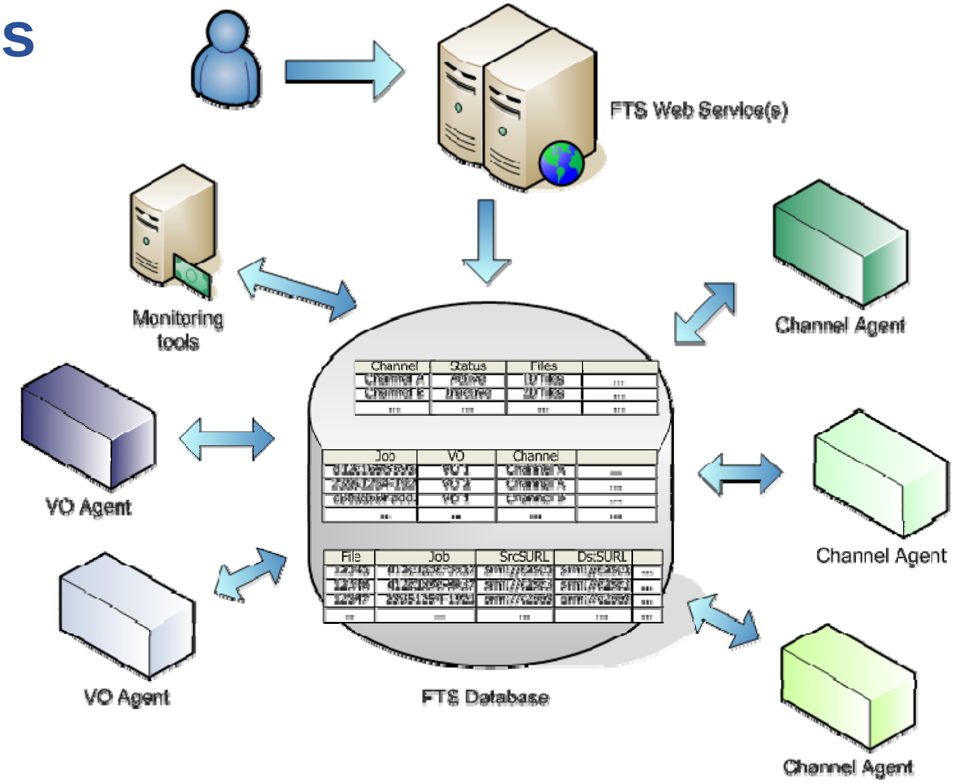
Goals

- **Provide a service**
  - Easy to manage and maintain
- **In a distributed (grid) environment**

- **Scalability**
- **High availability**
- **Robustness and resilience**
  - Run under unusual or stress conditions
  - Ability to recover from failures

Architecture

- **Decoupled components**
- **Web service**
  - Most critical component
  - Stateless
  - Easy to load-balance
- **Agents**
  - One per VO/channel
  - Split across multiple nodes
- **Clear responsibilities**
  - More easily identify problems



Channel	Status	Files	
Channel A	Active	10 files	...
Channel B	Inactive	20 files	...
...	...	...	...

Job	VO	Channel	
[illegible]	VO 1	Channel A	...
[illegible]	VO 2	Channel A	...
[illegible]	VO 1	Channel B	...
...	...	...	...

File	Job	SrcSURL	DstSURL	
12345	[illegible]	[illegible]	[illegible]	...
12346	[illegible]	[illegible]	[illegible]	...
12347	[illegible]	[illegible]	[illegible]	...
...	...	...	...	...

Focus on the service

- **Concentrate on how the software will be operated**
- **Design with procedures in mind**
  - What will happen during maintenance?
  - What's the impact of hardware and software failures?
- **Decoupled components**
  - Service is easier to maintain
  - Smaller impact of failures
  - Upgrade to SLC4 with zero user-visible downtime
  - Still to be improved
    - No "hot spares" for the agents
    - Moving (slowly) towards automatic failover and hot-standby
- **Configuration in the database**
  - Stop a node and restart it on another machine

Distributed environment

- **Design against unavailabilities and failures of**
  - sub-components of the service
  - other services

- **Add resilience to glitches**
  - Retry connections
  - Cache data locally

- **Encapsulate interaction with services**
  - layer over SRM 1.1 and 2.2

Design principles

- **Design from the beginning for robustness and resilience**
  - Discussed deployment and operational features have an impact on the basic architecture and design of the software
- **Retrofitting is expensive**
  - It is much harder to add high-availability features after the design and implementation (although it is possible).
  - See the agents’ “hot spares”

Schema design

- **Force integrity constraints on the DB**
  - Catch application logic errors that can otherwise be difficult to detect
  - Prevent logical schema corruption (extremely high cost on a production system)

- **Involve your database administrator in the design**
  - Use of bind variables
  - Appropriate use of indices
  - Table partitioning

Database interaction

- **Don’t treat the DB as a black box**
  - Use db specific features to improve performance
  - Transparent application failover (Oracle)
- **Use connection pools**
  - Standard connection pooling implementations available
  - Critical for performance (reduce number of connect / disconnect operations)
- **Connection retries**
  - Hide connectivity glitches
- **Data caching**
  - Cache frequently used and rather static information (channel definitions)

Testing

- **Unit tests**
  - Check not to break functionalities
  - Bug regressions
- **Functional tests**
  - Interaction of service with other services
    - SRM (transfer layer designed as a plugin).
- **Pilot service**
  - Test the whole service at an appropriate scale
  - Many issues only appear at close-to-production scale