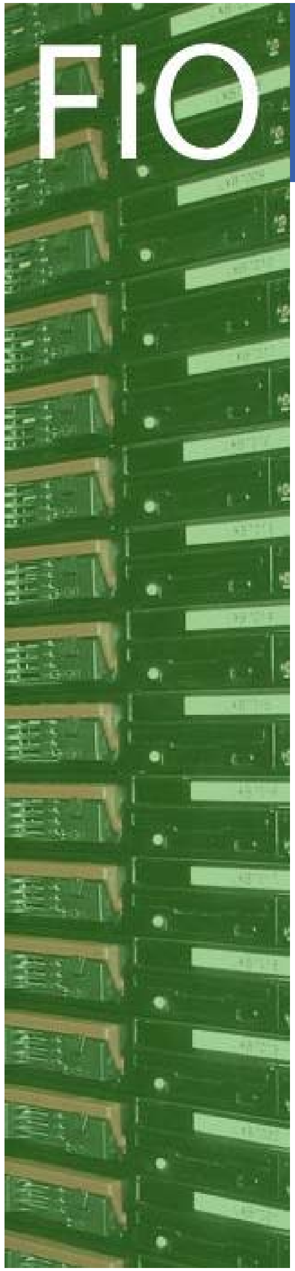


Fabric Infrastructure
and Operations

CERN IT
Department

Towards end-to-end debugging for data transfers

Gavin McCance
Javier Conejero Banon
Sophie Lemaitre
CERN IT/FIO

CHEP 2009

Outilne

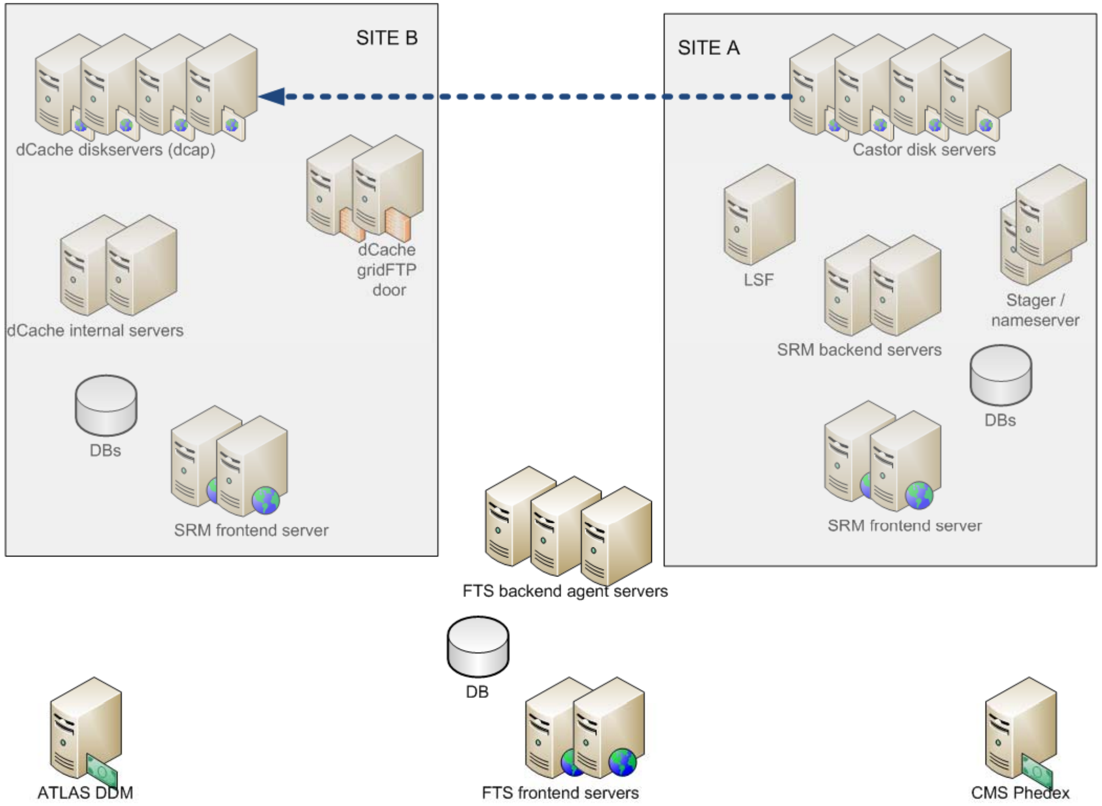
- The challenge
- Our problem
- Our solution

Outilne

- **The challenge**
- Our problem
- Our solution

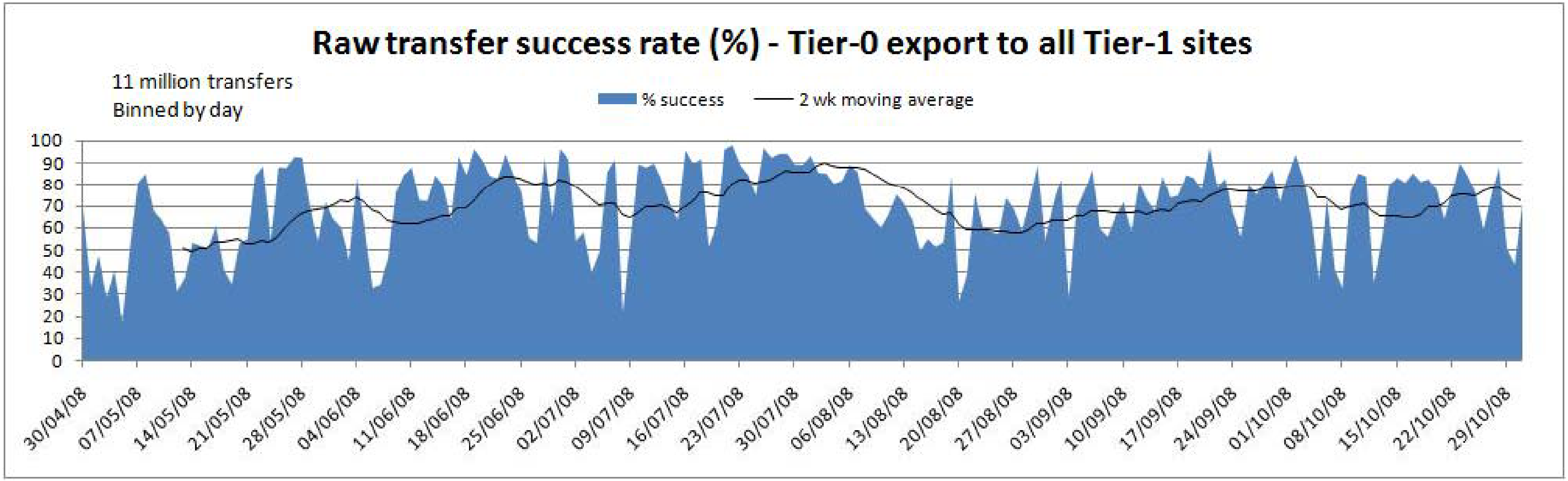
The challenge

- The data service model we have in the WLCG is all a bit complex
- There are many layers of software involved
  - Expt framework<->Transfers<->SRMs<->Stagers<->gridFTP<->Tape
- Individual components are quite internally complex



Efficiency?

- WLCG planning meeting November 2008:
- Examined efficiency of the whole data management stack
  - All the files get there in the end! (multiple retries)
  - RAW transfer rate (#successes / total # attempts, per day)
    - Failure can and do happen in any layer, at both ends of a transfer
- About ¼ of all transfer *attempts* fail due to storage errors ! ☹



Fabric Infrastructure and Operations

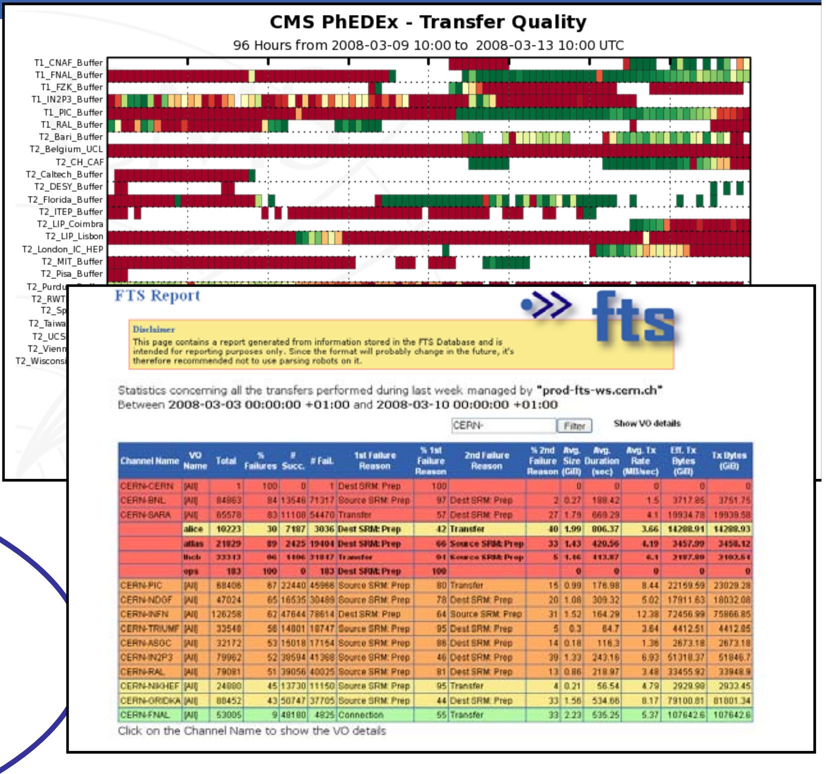
How can we improve this?

CERN IT Department

1. Summary “dashboards” collect ‘events’ and provides summary views, sliced in different ways
   - e.g. Current quality on transfers per site

2. Distributed debug tracing allows you to follow a specific operation through all the middleware components that process it
   - Show me the transfer for *this* file

- Service operations staff typically use
  - the first one to look for problems
  - second one to drill down and understand them



FTS Report

Disclaimer

This page contains a report generated from information stored in the FTS Database and is intended for reporting purposes only. Since the format will probably change in the future, it's therefore recommended not to use parsing robots on it.

Statistics concerning all the transfers performed during last week managed by "prod-fts-ws.cern.ch"
Between 2008-03-03 00:00:00 +01:00 and 2008-03-10 00:00:00 +01:00

CERN-

Show VO details

Channel Name	VO Name	Total	% Failures	# Succ.	# Fail.	1st Failure Reason	% 1st Failure Reason	2nd Failure Reason	% 2nd Failure Reason	Avg. Size (GiB)	Avg. Duration (sec)	Avg. Tx Rate (MB/sec)	Eff. Tx Bytes (GiB)	Tx Bytes (GiB)
CERN-CERN	[All]	1	100	0	1	Dest SRM: Prep	100			0	0	0	0	0
CERN-BNL	[All]	84863	84	13546	71317	Source SRM: Prep	97	Dest SRM: Prep	2	0.27	188.42	1.5	3717.85	3751.75
CERN-SARA	[All]	65578	83	11108	54470	Transfer	57	Dest SRM: Prep	27	1.79	669.29	4.1	19934.78	19939.58
	alice	10223	30	7187	3036	Dest SRM: Prep	42	Transfer	40	1.99	806.37	3.66	14288.91	14288.93
	atlas	21829	89	2425	19404	Dest SRM: Prep	66	Source SRM: Prep	33	1.43	420.56	4.19	3457.99	3458.12
	lhcb	33343	96	1496	31847	Transfer	91	Source SRM: Prep	5	1.46	413.87	4.1	2187.80	2192.54
	ops	183	100	0	183	Dest SRM: Prep	100			0	0	0	0	0
CERN-PIC	[All]	68406	67	22440	45966	Source SRM: Prep	80	Transfer	15	0.99	176.98	8.44	22159.59	23029.28
CERN-NDGF	[All]	47024	65	16535	30489	Source SRM: Prep	78	Dest SRM: Prep	20	1.08	309.32	5.02	17911.63	18032.08
CERN-INFN	[All]	126258	62	47644	78614	Dest SRM: Prep	64	Source SRM: Prep	31	1.52	164.29	12.38	72456.99	75866.85
CERN-TRIUMF	[All]	33548	58	14801	18747	Source SRM: Prep	95	Dest SRM: Prep	5	0.3	84.7	3.64	4412.51	4412.85
CERN-ASGC	[All]	32172	53	15018	17154	Source SRM: Prep	86	Dest SRM: Prep	14	0.18	116.3	1.36	2673.18	2673.18
CERN-IN2P3	[All]	79962	52	38594	41368	Source SRM: Prep	46	Dest SRM: Prep	39	1.33	243.16	6.93	51318.37	51846.7
CERN-RAL	[All]	79081	51	39056	40025	Source SRM: Prep	81	Dest SRM: Prep	13	0.86	218.97	3.48	33455.92	33948.9
CERN-NIKHEF	[All]	24880	45	13730	11150	Source SRM: Prep	95	Transfer	4	0.21	56.54	4.79	2929.98	2933.45
CERN-GRIDKA	[All]	88452	43	50747	37705	Source SRM: Prep	44	Dest SRM: Prep	33	1.56	534.66	8.17	79100.81	81001.34
CERN-FNAL	[All]	53005	9	48180	4825	Connection	55	Transfer	33	2.23	535.25	5.37	107642.6	107642.6

Click on the Channel Name to show the VO details

Outilne

- The challenge
- **Our problem**
- Our solution

Operational cost

- Operational cost of distributed debugging is still too high
- Currently it’s grep intensive over all across multiple services
  - wassh –l root –c gridfts grep reqID /var/tmp/*failed/glite*.log
  - wassh -l root -c gridsrm/atlas zgrep "49b4fa78-0000-1000-ad97-fa78af063a57" /var/spool/srm/log.4.gz
- This impacts sites and experiment shifters
  - Training curve is rather steep for new operations staff
  - Inaccessible files (~hours)
  - gridFTP mysteriously timing out (bouncing emails/phone calls back a couple of times to the other site) (~hours)
  - “We see reduced transfer rates, please could you check” (~hours)
  - Performance variance is typically very large and not well understood
    - Some files transfer at 10MB/s, some go a 200KB/s, same site, same time
- Better debug tools can reduce operations cost!

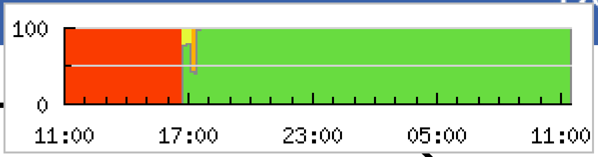
What we're aiming for

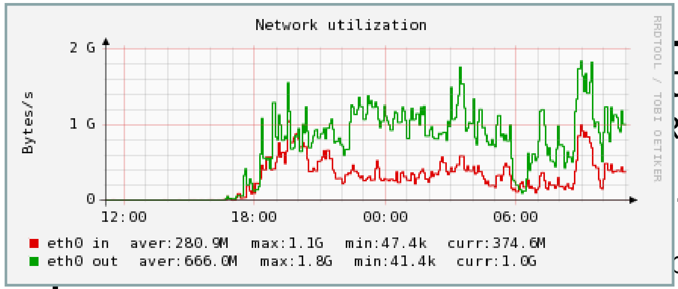
1. A support ticket comes in
   - "We see lots of transfers timing out"
   - Example file:
     - /castor/cern.ch/grid/atlas/atlasdatadisk/data08_cosmag/ESD/data08_cosmag.00090272.physics_RPCwBeam.reco.ESD.o4_r560_tid027478/ESD.027478._00769.pool.root.1"

2. Submit request to debug transfer for this file

3. Picture will be built up asynchronously as data is returned from the various sources, like a web-page loading

What we’re aiming for





transfer summary:

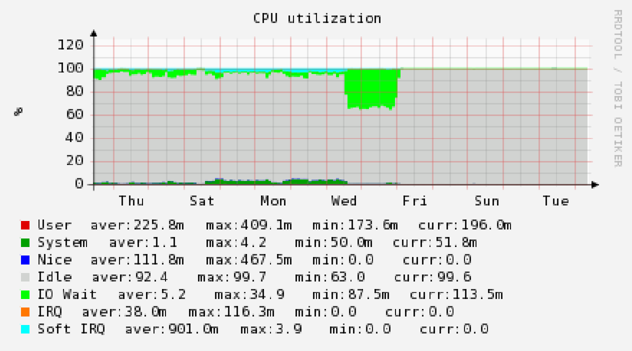
ROD, RAL-T1, atlas, “gridFTP: the server timed out”

Trace detail:

pareToGet -> CERN: detail

srmGetStatusOfGet -> CERN: detail

Srm gsiftp returned CERN: gsiftp://lxfsrc0203.c



SRM service: received call
Scheduled on stager
TURL determined

Stager: request schedule o

LSF scheduler: diskserve

srmPrepareToPut -> RAL: detail

srmGetStatusOfGet -> RAL: detail

Srm gsiftp returned RAL: gsiftp://dispool0023.r

SRM service: received call
Scheduled on stager
TURL determined

Stager: request schedule of job of diskserver

Scheduler: diskserver access scheduled

gridFTP 3rd party call:
CERN -> RAL: detail

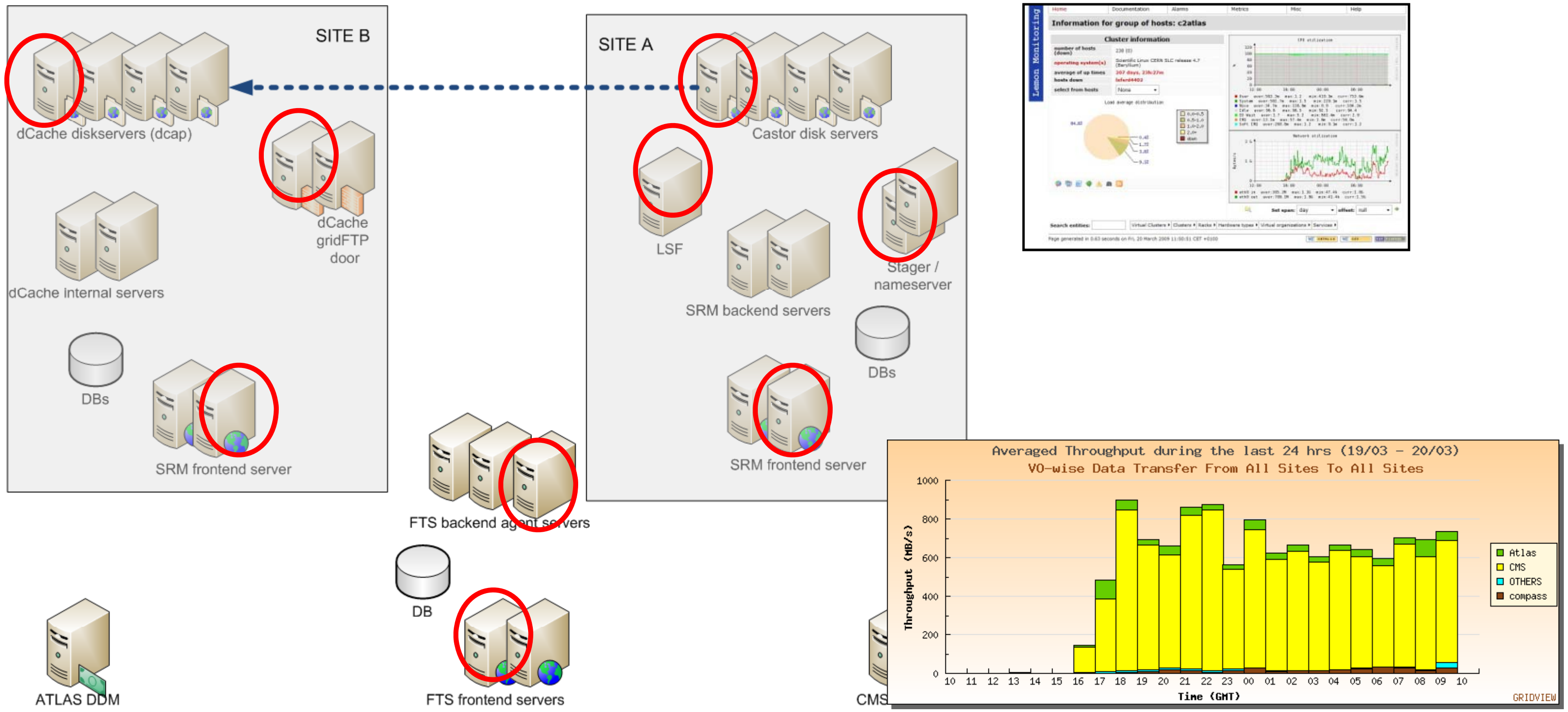
:

GridFTP RAL: FTS client connect
Opening data connection to other side on port X
Timeout!

:

Integration problem

- It's an integration problem



  - Multiple logfile / database / feed formats to be parsed
  - Logs located on multiple machines (O(1000) nodes @CERN)

Outilne

- The challenge
- Our problem
- **Our solution**

Previous attempts: parse it all

- Our previous attempts focused on recording *all events*
  - You collect all events from all sources, all the time, parse them, and put them in an index database
    - Specific debug searches can be run over the database
  - Approach used by Splunk
  - Common logging instrumentation: approach taken by netlogger

- While this does work, routine parsing, collecting and joining can be expensive
  - Parsing 10’s GB’s of logs from O(1000) machines
  - It’s overkill for this application
    - A typical service manager will probably run no more than 0(100) debug trace queries a day, and we know what queries will be run

- We prefer to parse on demand
  - Can make use of debug trace databases if they are available

Our approach

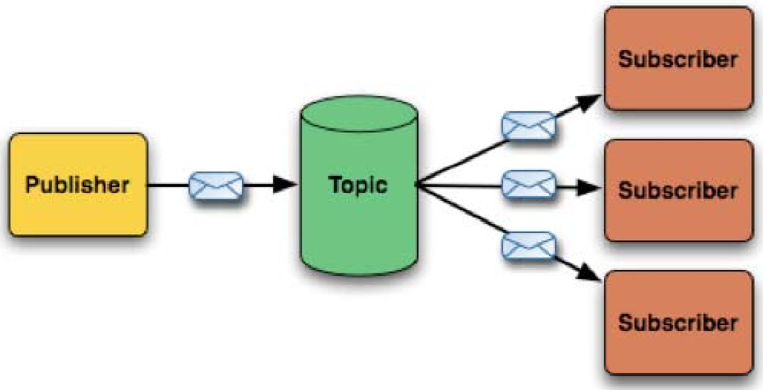
- On-demand extraction from data sources (request / response)
  - Send out requests to all data sources that might know something, get them to parse and return what they know
  - If sufficiently detailed summary or trace logging databases are available, use them
  - Integrate other feeds (fabric monitoring, network monitoring data)

- Integrate (join) the data from the various sources for that specific debug request
  - The flow is asynchronous, i.e. the picture of what happened is built up as information is returned
  - Even with missing information, the picture obtained is still useful for debugging

Architecture

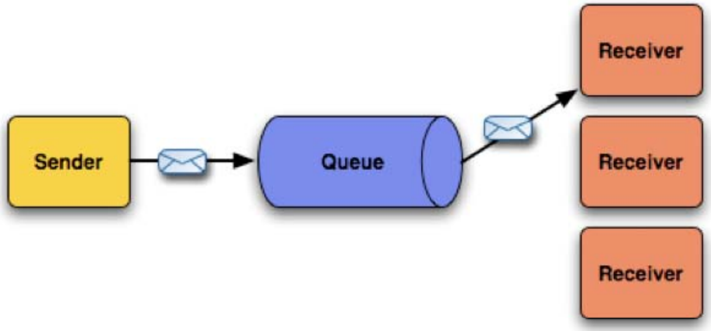
- Based on message-oriented middleware
- This handles the request / response reliably and easily

Publish-Subscribe / Broadcast



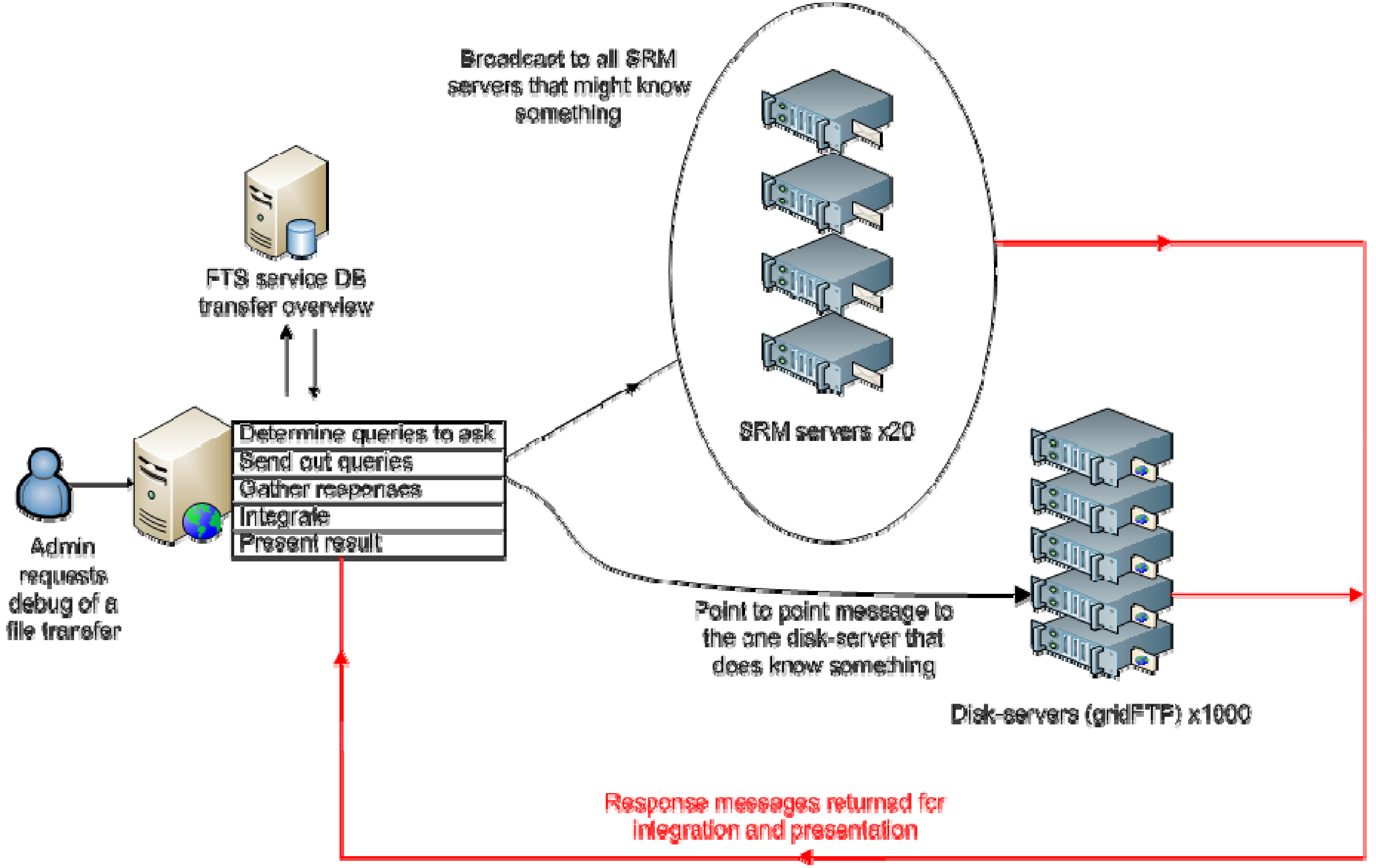
- Send a query to all nodes that might know something
- e.g. all SRM nodes in a load-balanced alias

Point-to-point



- Send a query to the one diskserver that we know handled the transfer
- e.g. gridFTP logs

Architecture


Broadcast to all SRM servers that might know something
FTS service DB transfer overview
SRM servers x20
Determine queries to ask
Send out queries
Gather responses
Integrate
Present result
Admin requests debug of a file transfer
Point to point message to the one disk-server that does know something
Disk-servers (gridFTP) x1000
Response messages returned for integration and presentation

- The message system handles the plumbing and the reliable delivery of the messages – local agents do the parsing

Messaging technology

- Using the MSG messaging framework
  - Technology already used in WLCG production in EGEE/OSG for grid site-monitoring data
  - See EGEE User Forum for details of MSG:
  - http://indico.cern.ch/contributionDisplay.py?contribId=136&sessionId=9&confId=40435
  - Uses Apache ActiveMQ: open source, easy to use

- ✓ Throughput requirements
  - Isn't really an issue for administrator initiated requests: O(100) per day
- ✓ Latency requirements
  - Needs to deliver fast – we don't want to be waiting too long
- ✓ Reliability requirements
  - We do care that the messages get there in order to build up a full picture
- ✓ Scaling requirements
  - We need it to scale up to O(1000) nodes so that we can run this over all our diskservers

Current status

- Planning to integrate all data at CERN from:
  - File Transfer Service (Tier-0 physics data export service)
  - Castor SRM and Castor core components
  - Lemon fabric monitoring service
- Aim: tool usable by service managers in summer to help with the transfer debugging problem
- Future:
  - Add data feeds from other sites (other SRMs): collaboration with external sites. Add network monitoring data
  - Tool itself useful for other sites?
  - Re-use components for distributed workload-management services?

Flexible architecture

- Future re-plumbing is easy: the architecture allows us to easily change the data sources as software develops
  - *Decide we want to collect and archive gridFTP logs on 10 central machines*
    - Move the gridFTP agents off all your diskservers to just these 10 machines instead, to answer the same request
    - The rest of the system remains unchanged
  - *Next version of one component comes with a detailed-enough trace database?*
    - Unplug all the log-mining agents and plug on an agent to answer the same request from the trace database instead
    - The rest of the system remains unchanged
  - *Want to add in network flow data ?*
    - Write another feed to make this data available and add it in

Summary

✓ Aim: to reduce operations cost of running complex distributed services

- Developing a flexible architecture based on messaging for trace-debugging of distributed services
  - Parse logs as data sources
  - Use trace database sources if available
- Integrate data on-demand from various sources instead of routine parsing

✓ Will have a usable tool to help with the transfer debugging problem by summer

Backup

Previous attempts: common formats

- Using common formats and even better a common logging trace schema for all components involved is a great idea!

- Easier to do if you control all the components
  - e.g. most components of Castor drop trace info into a distributed tracing component (DLF database)
  - *Netlogger* calls can be added to the code to send data streams out
- Hard for other components
  - Some bits of the code we don't 'own' (Castor: *LSF, gridFTP*), so it can be hard to add trace info at the level needed
  - Why should FTS, dCache, Lemon, Nagios log into the same format?

- While this is a good goal we prefer to deal with the integration problem we have directly