Resilient FTS3 service at GridKa

T. Hartmann, J. Bubeliene, B. Hoef, L. Obholz, A. Petzold, K. Wisniewski
Karlsruhe Institute of Technology (KIT), Steinbuch Centre for Computing (SCC), Hermann-von-Helmholtz-Platz 1, D-76344 Eggenstein-Leopoldshafen, Germany
E-mail: thomas.hartmann@kit.edu

Abstract. The FTS (File Transfer Service) service provides a transfer job scheduler to distribute and replicate vast amounts of data over the heterogeneous WLCG infrastructures. Compared to the channel model of the previous versions, the most recent version of FTS simplifies and improves the flexibility of the service while reducing the load to the service components. The improvements allow to handle a higher number of transfers with a single FTS3 setup. Covering now continent-wide transfers compared to the previous version, whose installations handled only transfers within specific clouds, a resilient system becomes even more necessary with the increased number of depending users.

Having set up a FTS3 services at the German T1 site GridKa at KIT in Karlsruhe, we present our experiences on the preparations for a high-availability FTS3 service. Trying to avoid single points of failure, we rely on a database cluster as fault tolerant data back-end and the FTS3 service deployed on an own cluster setup to provide a resilient infrastructure for the users. With the database cluster providing a basic resilience for the data back-end, we ensure on the FTS3 service level a consistent and reliable database access through a proxy solution. On each FTS3 node a HAproxy instance is monitoring the integrity of each database node and distributes database queries over the whole cluster for load balancing during normal operations; in case of a broken database node, the proxy excludes it transparently to the local FTS3 service. The FTS3 service itself consists of a main and a backup instance, which takes over the identity of the main instance, i.e., IP, in case of an error using a CTDB (Cluster Trivial Database) infrastructure offering clients a consistent service.

1. Introduction

The Grid Computing Centre Karlsruhe (GridKa)[1] at the Steinbuch Centre for Computing (SCC) within the Karlsruhe Institute of Technology (KIT) provides computing resources to a large number of users within the high energy physics community. Within the Worldwide LHC Computing Grid (WLCG) it serves as the German tier-1 center to all major LHC experiments with further HEP (high-energy physics) and astrophysical collaborations outside the WLCG using resources at GridKa. With the large number of heterogeneous users, the offered resources and services have to be robust and resilient due to the broad impact a service outage would cause.

We explore the setup of a File Transfer Service (FTS) instance as a resilient stack, consisting of two dynamic-master clusters. The previous version FTS2 was designed to broker large scale file transfers within the initial hierarchical tier structure of the WLCG. Its successor FTS3 has become more flexible and light-weight with managing transfers not bound anymore on a strict tier scheme and permitting data flows between all sites [2]. This allows to reduce the number
The FTS3 stack (see Fig. 1) consists of two clusters with no dedicated master-slave hierarchy. For one, a database cluster provides a resilient database and operates with a multi-master topology where all nodes have equal roles. The database cluster consists of three dedicated machines hosting a MySQL database as a Galera cluster and stores the FTS3 transfer states. Database transactions send to one node are replicated synchronously to other cluster members, thus no conflicts can arise as it would be in case of an asynchronous replication. The second cluster covers the FTS3 service, where the cluster nodes dynamically arbitrate between themselves a temporary topology. The FTS3 cluster consists of two dedicated machines, with one dynamically chosen master node and a second failover node as stand-by. Using dynamic clusters, we reduce single points of failures and gain robustness to outages of individual components. The general setup is not specific to the described FTS3 service and can be adapted to other services as well with features as load-balancing in addition to the failover mechanism. If necessary, both clusters can be extended with additional dedicated or virtualized nodes.

2. Hardware Setup
For each cluster its members are distributed over more than one rack to be operational in case of a rack-wide outage. Shared resources by the nodes are infrastructure components as shared storage and the network infrastructure.
The FTS3 cluster consists of two physical machines, each one connected to the network by two gigabit network cards. One network card is connected to the general purpose internet and provides the interface for the users of the FTS service. The second network card is connected to the GridKa internal production network for database access, monitoring and shared storage.

The Galera MySQL cluster consists of three physical machines equipped with sufficient memory and CPUs. Each node has one network card connected to the internal network, thus separated from the general purpose internet.

The network topology is sketched in Fig. 2. The central backbone of the KIT network consists of four central routers, which are connected in a ring topology and can tolerate an outage of one element. The router mesh is connected to a firewall instance, which consists of two elements in a failover configuration. The firewall’s border routers provide links to the regional and national National Research and Education Networks (NREN) as well as LHCONE and LHCOPN [4].

Each FTS3 node is connected to one hardware switch with both nodes attached to different switches for redundancy. Each hardware switch manages the routes to the internal and external networks as two virtual switch instances. The database nodes are connected to one switch. For redundancy the switch itself is connected to two central routers. With all database nodes located at the same network leaf, operability depends crucial on the switch availability. We decided to move the database nodes close within the network topology, since the cluster nodes ensure the database integrity by a synchronous replication certifying each commit. Thus, the database performance depends on the latency within the cluster. A shared storage is located on a GPFS file system and is exported via the NFS protocol.

3. FTS3 Cluster Setup

We abstract the FTS3 service via virtual IP addresses. In addition to static IP addresses, nodes are assigned virtual IP addresses. Here, the dynamically allocation of a virtual IP is organized internally by the members of the FTS3 cluster. An overview of the FTS3 cluster setup is shown in Fig. 3.

We implemented the setup with CTDB [5, 6], which has been developed as cluster solution for SAMBA/CIFS. A set of virtual IP addresses is allocated to the CTDB cluster. CTDB daemons running on each node arbitrate dynamically a master instance that assigns the virtual IPs to the cluster members. If the current master drops out of the cluster, the remaining nodes will again arbitrate a new master. If any member of the cluster with virtual IP addresses assigned drops out, its virtual IPs will be re-assigned by the master to active members. As a separate lock device, a shared network file system is mounted on all cluster nodes hosting a shared file. Configuration files, that are the same on all nodes, are kept on the shared storage as well, e.g., a list of all cluster members’ internal IP addresses and the list of virtual IP addresses.

Since we use NFS as network file system, which uses only user ID numbers (UID) and group ID numbers (GID) for permission management, we harmonized the UIDs and GIDs over all FTS3 nodes.

We use two virtual public IP addresses: one as official FTS3 service address and one as maintenance address for the standby node. Due to the dynamic allocation, each of the two FTS3 nodes can in principle become the host of the FTS3 service IP with the other node becoming the standby node. A DNS record is assigned to the official service IP, i.e., \texttt{fts3-kit.gridka.de} and promoted to the FTS3 users.

Due to the virtual IPs and their assigned DNS records, each FTS3 cluster nodes’ host certificates have to contain all DNS names assigned to the cluster’s IP pool.

For easier debugging, we tune the CTDB cluster to try to pin a shared IP address to a node and reduce address migration between nodes. Also virtual IP addresses are not re-assigned, if all members of the cluster become not productive. Thus, if the whole cluster is unhealthy, any
Figure 3. Clustered FTS3 instances sharing a virtual IP address. Cluster members communicate with each other via CTDB and dynamically allocate virtual IP addresses between each other. For enhanced reliability a shared file system provides a common file locking path.

Figure 4. On each FTS3 node, database access is directed to a local HAProxy instance. The HAProxy serves as a round robin and distributes queries over the members of the database cluster. Every HAProxy instance checks regularly each database node’s mysql health status and the status of the node in the Galera cluster.

virtual IPs are released until at least one member of the cluster becomes available. This setups allows a transparent way to update individual FTS3 nodes by successively dropping nodes from the cluster, applying updates and joining the cluster again. Similar, system updates on database nodes can be applied in a round-robin way as well. Exceptions are updates requiring changes to the database schema, which need the whole database cluster to be stopped and thus the FTS3 service as its client.

For central monitoring, a Icinga instance collects monitoring information and sends a notification to the service experts if a service degrades. For each node, general health states, e.g., on the network response or free space, are collected from each node. For monitoring the FTS3 service, we re-use tests from our FTS2 service. FTS3 test transfers and the FTS3 web service’s availability are checked regularly on each cluster member as well as on the shared IP/DNS. Additionally, the integrity status of each CTDB daemon is tested.

In our current setup, CTDB serves as failover mechanism moving the FTS3 service identity to the standby node, if the active node fails. For other use cases, CTDB could also be deployed in more complex setup, since the numbers of nodes or virtual IP can easily be increased. For example, as load-balancing cluster, CTDB can handle a larger set of IP addresses and distributing them dynamically over the nodes so that a node can assume multiple identities. All virtual IP addresses could be subsumed in one DNS record and clients could be distributed over the CTDB cluster member by a round-robin or similar scheduler and thus could provide load-balancing in addition to failover.

Since the expected load on a FTS node is much reduced in FTS3 compared to FTS2, we decide against load-balancing and use one single name per DNS record. This allows simpler management and debugging while still profiting from a failover mechanism.
4. Database Setup

4.1. DB Access

Requests to the database can be handled equally by all nodes of the Galera cluster. Galera [9] synchronously replicates changes over all cluster members using a transaction certification mechanism, thus conflicts are avoided that can arise in asynchronous replications as in standard versions of MySQL or MariaDB. One possible draw-back of Galera’s opportunistic locking can occur with conflicting write requests, with heavier load on the database than a pessimistic locking due to increasing probability of transaction rollbacks for database consistency [10]. In contrast, if the majority of requests are read transactions, optimistic locking allows read requests to be distributed in parallel over all nodes.

We balance read and write transactions uniformly over all cluster nodes, since we do not expect the frequency of commits to become large enough to suffer from performance losses with to Galera’s optimistic locking. An intermediate HAProxy [11] on each FTS3 node balances write/read requests in a generic tcp mode over all database nodes as sketched in Fig. 4. DB queries from a node’s FTS3 instance are directed to localhost with the HAProxy instance further redirecting queries with a round-robin scheduler to one of the Galera cluster members.

4.2. DB Performance

With two layers between the application and the database, we study the performance impact from the Galera clustering and the intermediate proxy by measuring the number of transactions per second through the later. The direct database performance is measured on each node with cluster replication deactivated also excluding network latencies. Following, we benchmark the database performance on a node in an active cluster giving the replication impact. For network latency performance-losses, database throughput is measured on the FTS3 nodes to each database nodes in the cluster. Finally, we measure the performance on a FTS3 node through an active HAProxy including performance-losses due to load-balancing. We use from sysbench [12] the read-only and complex MySQL benchmark options to differentiate between plain read and complex write transactions for potential replication effects. Fig. 5 shows the database performance in the FTS3 stack.

As expected, in all layers more write transactions can be performed per time interval than complex transactions. While the performance decreases by $\sim 27\%$ for complex transactions with replication being active, the averaged number of read transactions increases unexpectedly by $\sim 61\%$ with benchmarks performed on each database node directly.

Accessing from a FTS3 node over the network the database cluster with active replication reduces the throughput by $\sim 45\%$. The second FTS3 node performs slightly better than the first node, which we attribute to the more powerful hardware and its broader network connection. With database queries distributed by HAProxy, we observe for the first FTS3 node a performance decreases by $13\%-18\%$ and for the second node of $3\%-8\%$, benefiting from the more powerful CPU.

4.3. DB Node Monitoring

All database nodes are monitored centrally by the Icinga instance. On each database node, scripts, available as xinited services, test the response of the local mysql service as well as the Galera cluster integrity and the size of the Galera cluster as known to this node. If a member of the database cluster fails, queries from a FTS3 node would also be at risk to be directed by HAProxy to the broken database node. Thus, we setup HAProxy to check frequently the health of each node of the Galera cluster. We re-use the checks for Icinga and pass the basic check results into a wrapper. The wrapper, also available as xinited service on each node, generates a http 200 OK success status for a running service and a http 500 Internal Server Error error status for failures. With a background as http load balance, HAProxy can interpret these http
status codes. Thus, HAProxy excludes a node if a test returns an error. An excluded node is re-enabled when three successive tests are successful. We enable HAProxy to publish an HTML page within the local network showing statistics as uptimes, errors or transferred data.

![Database performance measured over successive software and network layers. Averaged read-only/complex write transactions per second measured: each database node without and with replication, from each FTS3 node without/with intermediate HAProxy over the network with active replication.](image)

**Figure 5.**

### 5. Conclusion

We present a FTS3 setup aimed for resiliency of its two core components. Both, the database back end as well as the FTS3 service, are clusters that can withstand the loss of machines with failover mechanisms transparently for clients. Both clusters can be scaled by adding further nodes. As general solution for IP address sharing and allocation, CTDB can be used for other services as well. HAProxy provides a round-robin scheduler for distributing database queries safeguarded with integrity checks of the MySQL cluster. The Galera database cluster avoids collisions by a certification based replication.

Possible improvements or changes on the clusters or the services are conceivable:

- Using the shared file system for storing transfers log files. When supported, transfer log files can be served to users through the FTS3 monitoring interface independently from the node managing the actual transfer. We plan to implement it when support for dedicated log paths becomes available.

- CTDB can be extended with event scripts to perform actions during status changes of the cluster or for regular monitoring. Thus, active monitoring could be moved from external checks by HAProxy or Icinga into state driven events from within the CTDB cluster.

- Moving a database node into another network segment for better resilience. This will require further benchmarking of the database performance and latency due to the inter-node replication.

- Integrating the database cluster into a CTDB setup with shared IP as for FTS3. Instead of using HAProxy on each FTS3 node, the database cluster could be provided via CTDB as a unified resource to the FTS3 service. This would require to integrate the CTDB mechanisms with the Galera high-availability service and to reconsider a load distribution solution if becoming necessary.

- Targeting database queries to one database node and failover to other database nodes only if the primary node fails its health checks.

- If the load due to colliding transactions would become to large, HAProxy could be configured to distribute read operations over all nodes while directing write commits to one dedicated node and, thus, optimize the cluster load due to minimized synchronous...
replication locking[13]. However, this would require the FTS3 application to support different ports for read and write operations.

- Setup of a fencing mechanism to securely remove broken nodes from the cluster. E.g., an integration with Pacemaker/Linux-HA[14, 15]. Since both systems, CTDB and Pacemaker provide partly overlapping functionalities, a careful configuration would be necessary to integrate the system-specific parts from each solution with each other. Since the FTS service is essentially stateless and the database contains all transfer information, an aggressive fencing mechanism is not essential and would not necessarily increases the reliability of a FTS3 setup.

References
[1] GridKa (Grid Computing Centre Karlsruhe), http://www.gridka.de
[2] A A Ayllon and M Salichos and M K Simon and O Keeble FTS3: New Data Movement Service For WLCG, J. Phys.: Conf. Ser. 2014, 513 3 032081, doi:10.1088/1742-6596/513/3/032081, http://dx.doi.org/10.1088/1742-6596/513/3/032081,
[3] Codership Oy. Galera cluster for MySQL the true multi-master, http://galeracluster.com/products/
[4] LHCOPN Working Group Large Hadron Collider Optical Private Network, LHC Open Network Environment, https://twiki.cern.ch/twiki/bin/view/LHCOJP/WebHome/
[5] CTDB cluster implementation of the SAMBA trivial database (TDB), https://ctdb.samba.org/
[6] CTDB project wiki, https://wiki.samba.org/index.php/CTDB_Project
[7] Icinga Open Source Monitoring, https://www.icinga.org/
[8] dCache.org dCache, a distributed data storage caching system, http://www.dcache.org
[9] Codership Oy. Galera database replication documentation, http://galeracluster.com/documentation-webpages/introduction.html
[10] severalnines Limitations in Galera Cluster for MySQL, http://support.severalnines.com/entries/21692388-Limitations-in-Galera-Cluster-for-MySQL
[11] HAPProxy The Reliable, High Performance TCP/HTTP Load Balancer, http://www.haproxy.org/
[12] sysbench https://github.com/akopytov/sysbench
[13] severalnines Avoiding Deadlocks in Galera - Set up HAPProxy for single-node writes and multi-node reads, http://www.severalnines.com/blog/avoiding-deadlocks-galera-set-haproxy-single-node-writes-and-multi-node-reads
[14] Linux-HA Providing Open Source High-Availability Software for Linux and other Platforms since 1999, http://www.linux-ha.org/wiki/Main_Page/wiki/Pacemaker
[15] Corosync alternative cluster communication managern (CMM) to Heartbeat, https://corosync.github.io/corosync/