Development of an Innovative Storage Manager for a Distributed Control System

M. Mara INFN-AC, Frascati, Italy
C. Bisegni, G. Di Pirro, L.G. Foggetta, G. Mazzitelli, A. Stecchi, INFN-LNF, Frascati, Italy
L. Catani, INFN-Roma2, Rome, Italy

ICALEPCS 2013
TUPPC011

!CHAOS Control system based on a Highly Abstracted and Open Structure

ABSTRACT

!CHAOS is an INFN project aimed at the definition of a new control system standard for large experimental apparatus and particle accelerators based on innovative communication framework and control services concepts. !CHAOS has been developed to address the challenging requirements in terms of data throughput of the new accelerators under study at INFN. One of the main components of the !CHAOS framework is the historical engine (HST Engine), a cloud-like environment optimized for the fast storage of large amount of data produced by the control system's devices and services (I/O channels, alerts, commands, events, etc.), each with its own storage and aging rule. The HST subsystem is designed to be highly customizable, such to adapt to any desirable data storage technologies, database architecture, or indexing strategy and fully scalable in each part. The architecture of HST Engine and the results of preliminary tests for the evaluation of its performance are presented in this paper.

THE !CHAOS FRAMEWORK

The !CHAOS framework has been designed after an in- depth evaluation of the new software technologies for data transfer and data storage emerging from the development of high-performance Internet services, such as the non-relational databases (NRDB) and the distributed caching system (DCS). Both are designed for a high degree of horizontal scaling that allows the insertion and retrieval of the data at the highest possible throughput, limited only by the saturation of either the available bandwidth or the network connections of the subsystem.
While the NRDB logics and techniques are used to implement the indexes management and the fast data retrieval the DCS is used to provide the "live data sharing", a scalable service for sharing the real-time device data. This software provides in-memory key/value storage and permits fast accesses to the same key/value by many concurrent clients. This caching layer avoids overloading the front-end controller with multiple reading accesses from clients that need to fetch data of a device.
These two software technologies represent the core components in the design of the new control system named !CHAOS [1, 2, 3].

THE !CHAOS STORAGE ENGINE

In !CHAOS the data storage is provided by the service called History (HST) Engine. This conceptual design will allow an innovative storage system for a Distributed Control System, giving !CHAOS an important technology advantage against other equivalent most popular standard for controls. The main ideas at the base of the data acquisition process are the following: a distributed file system is used to store data produced by machine operations while a KVDB manages the indexes structure (nowadays candidates are Hadoop [7] and MongoDB [8] respectively). These tools have been chosen thanks to their diffusion in the scientific community for solving similar problems and the abundance of use cases to which learn from. The functionalities of the !CHAOS HST Engine are allocated to three dedicated components, or nodes, namely the !CHAOS Query Language (CQL) Proxy, the Indexer and the Storage Manager.

Thanks to the chosen implementation, it is possible to increase the overall performance of the system by scaling different components. A faster writing mechanism for the devices can be ensured by increasing the number of proxies writing in a parallel way inside the cache. The transfer process between cache and device logical files can be increased by incrementing the number of managers checking the packets acquired; the indexing procedure can be improved by increasing the number of indexer nodes.

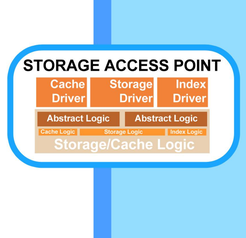
The Data Flow

By checking the medium data-pack size of the devices, and its pushing rate, the registered CQL Proxies are "allocated" to the devices in order to load balance the network infrastructure and the total computing power.

STORAGE ACCESS POINT

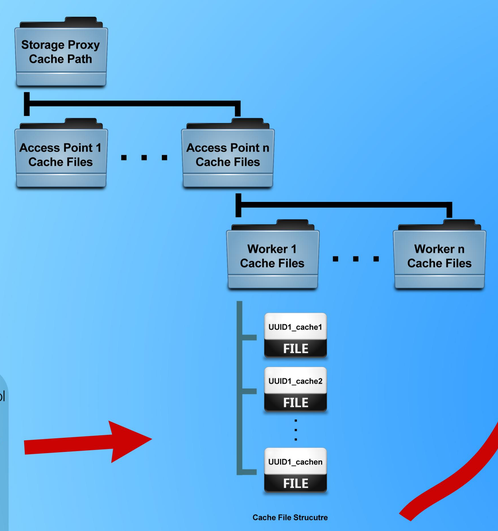
Storage Subsystem

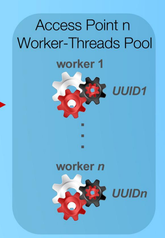
A Front End Controller, i.e. a !CHAOS Control Unit (CU), starts the writing process by pushing a dataset to one of the storage subsystem CQL Proxies.



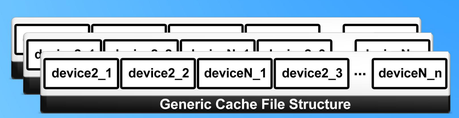
STORAGE ACCESS POINT

For each CQL Proxy a logical path is created in the distributed FS. To improve the performance of the system each proxy can allocate a pool of threads with the only task of getting the packets received by the proxy and start the allocation inside the file system.

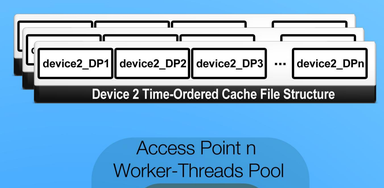


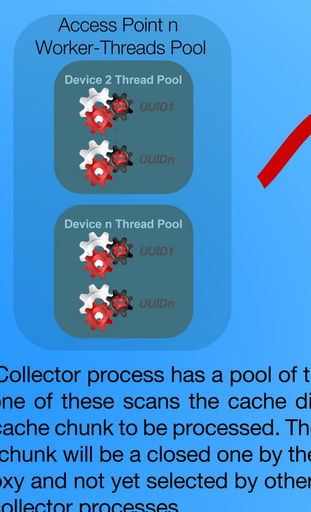


Each thread will fetch the data packets from the proxy regardless the device that produced them, starting to write them in files inside the path associated to the Proxy.



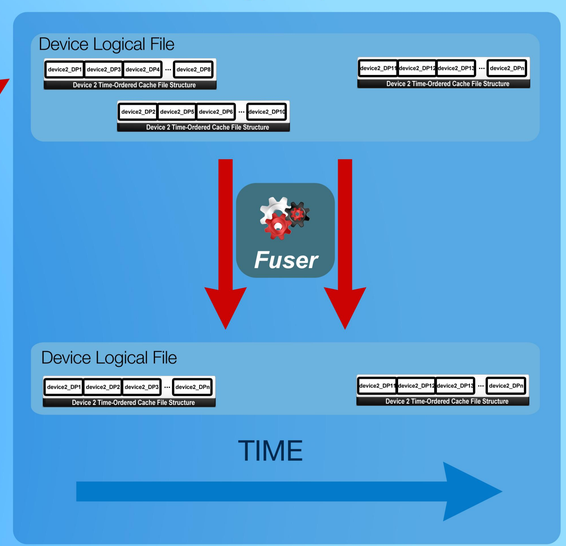
The data pack wrote by a thread is stored into a private cache chunk (CC). Once a CC is no more valid (in terms of space or time elapsed), it will be available for the next phase described below.

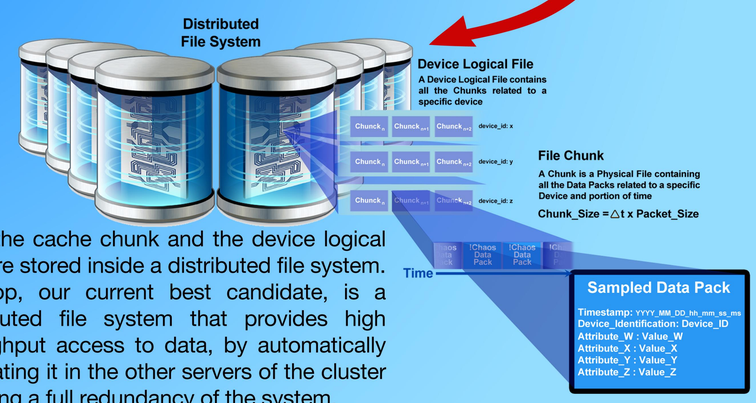




Every Collector process has a pool of thread and each one of these scans the cache directory to find a cache chunk to be processed. The selected cache chunk will be a closed one by the action of the proxy and not yet selected by other thread or other collector processes.

After the "Moving" phase (Figure 3), every chunk of the logical file will be ordered in time (every data packet has timestamp >= than the previous one). Anyway two or more LF chunks can be overlapped in time (a side effect of the fast moving phase).





Both the cache chunk and the device logical files are stored inside a distributed file system. Hadoop, our current best candidate, is a distributed file system that provides high throughput access to data, by automatically replicating it in the other servers of the cluster ensuring a full redundancy of the system.

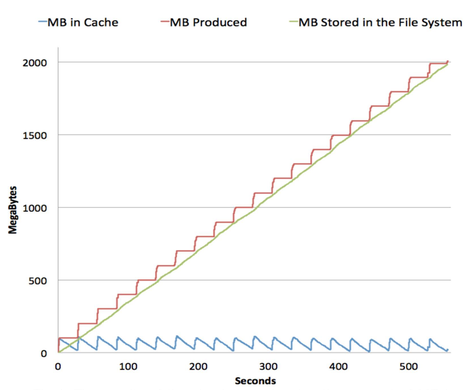


Fig 1: The results of the simulation, showing the amount of MB in cache, produced and stored in the file system.

TEST RESULTS

The components described so far, related to the staging and moving mechanism, have been tested off-line by using a software simulation of these two phases. The fusing phase has been removed from the numeric tests because is not fundamental for the data acquisition process: it is used once the data is already safe on the file system. The tests have been run on a mid level Mac Pro with two 2,8Ghz Quad-Core Intel Xeon, 18GB of DDR2 RAM, and a SA T A2 SSD hard disk. The graph shown here (fig. 4) is obtained by using two Producer processes simulating ten devices running with 50 threads each, and a single consumer process running on five threads. The average data produced by the simulated devices is 3,5 MB/s simulating 515 channels pushing data packets of 68 B at 100Hz. The test environment is like a worst-case scenario for this algorithm, because it cannot gain performance by a distributed file system and a multitude of proxy machines. In fact the data rates obtained can grow almost linearly by increasing the number of proxy machines and using a more appropriate file system. The graph in figure 4 shows the three fundamentals measurement in the caching system: the data produced by the devices (in red), the data actually in the cache files (in blue) and the data actually stored inside the device logical files (in green). More intensive tests will be run in the next months on the other parts of the storage system now under development.

REFERENCES

[1] http://chaos.infn.it
[2] L. Catani et.al., "Introducing a new paradigm for accelerators and large experimental apparatus control systems", Phys. Rev. ST Accel. Beams 15, 112804 (2012).
[3] L. Foggetta et.al., "Progresses on !CHAOS development", Proceedings of IPAC2012, New Orleans US, http://www.JACoW.org
[4] http://bsonspec.org
[5] http://hadoop.apache.org [6] http://www.mongodb.org

http://chaos.infn.it

Take Part!!

Common Integration System:
http://cvs.lnf.infn.it:8080

Institutional Git Repository (Public Read Access)
https://cvs:lnf.infn.it:8443/chaosframework.git

Public Git Repositories:
https://chaosframework.atlassian.net