dataClay: towards usable and shareable storage

Toni Cortes, Anna Queralt, Jonathan Martí, and Jesús Labarta

Barcelona Supercomputing Center (BSC-CNS)

As we have already presented in other BDEC papers [1][2], BSC believes that integrating the programming model and the management of persistent data is key to achieve high-performance applications, as well as to create these applications in a productive way.

This white paper can be seen as an extension of [1] and [2] by presenting dataClay, **a solution to manage persistent data** that addresses the following 4 concerns:

1. Effective use of new storage abstractions
2. Bridging the gap between storage and computation
3. Productivity when creating efficient code
4. Data sharing

In order for exascale and/or big-data systems to deliver the needed IO performance, new storage devices such as NVRAM or Storage Class Memories (SCM) need to be included into the storage/memory hierarchy. Given that the nature of these new devices will be closer to memory than to storage (low latencies, high bandwidth, and byte-addressable interface) using them as block devices for a file system does not seem to be the best option. With dataClay, we propose a **new generation of object storage** where objects (understood as in the object-oriented programing paradigm) are the persistent abstraction, as opposed to files, or databases. This persistent abstraction will enable both the programmer, and dataClay, **to take full advantage of the coming high-performance and byte-addressable storage devices**. Today, given the lack of such devices, dataClay performs a mapping of such abstractions to key-value stores such as Kinetic drives from Seagate.

DataClay **stores object data**, **methods** (code to be used to use the object) and **behavior policies** that define issues such as security, privacy, integrity, and life cycle among others. This differs from other approaches that offer objects as the persistent abstraction, such as object-relational mapping libraries or OODB that only store the object data. This integration of data and code enables, in a very simple way, to perform computation close to the data, thus **enabling the programmer or the middleware to decide where a given computation has to be executed**. And given that methods and behavior policies are part of the object, the object can be moved to different infrastructures and still behave as it was designed.

Another important feature of dataClay is the **tight integration with the StarSs parallel programming model** (currently COMPSs [3] and PyCOMPSs [2]). This integration enables persistent data to become a first class citizen. On the one hand, the programmer only needs to devise a **single data model that will work both for in-memory and for stored data**. This approach differs from current options where data in memory is handled by objects or abstract data types, and data in disk is kept in files or databases using a completely different

data model. This unified model has two big advantages. First, it **enables features to improve the performance of parallel applications**. For instance, we can implement persistent collections where the iterator is aware of locality, and thus deliver to each slave in the parallel application only the items that are close to it. Second, it enables programmers to be **more productive** by unifying the data model and thus removing the consequent model translation out of their programs.

Finally, the **value of data increases when it is shared**. Unfortunately, today's methods to share data are not optimal. Today we have three basic mechanisms to share data. First, we can share the data by giving access to the different actors to the shared data; this option is very flexible but also dangerous because the owner of the data cannot limit what can be done to the data in a fine grain way. Second, we can copy the data from the owner infrastructure to the infrastructure of the consumer; this option is also very flexible but incurs in a few problems such as unnecessary data copies, dificultes to keep the data up to date, and most importantly the loss of control of the data owner over its data. And third, we can encapsulate the data in a data service where the data owner implements what can be done with the data, and the rest of actors interact with the data using the existing options. This last approach has the problem that new operations on data need to be implemented by the data owner, who may not have the resources or interest to implement it.

In order to improve data sharing, **dataClay offers a mechanism to enrich data by third parties** (not the data owner) **without removing the control from the data owner**. By enrichment we understand the modification of the data model (adding new classes, or fields to existing classes), the attachment of new code to the data (adding new methods, new versions of existing methods, or even new behavior policies), and of course, the addition of new data objects to the data set. For instance, this functionality enables researchers to adapt data from other research groups without unnecessary duplications and without taking the control of the data from the data owner.

Summarizing, dataClay is platform ready to exploit the advantages of new storage devices, that is well integrated with the programming model (StarSs) increasing the application performance as well as the programmer productivity, and that enables much more flexible and controlled data sharing.

---

[1] Jesus Labarta, Eduard Ayguade, Fabrizio Gagliardi, Rosa M. Badia, Toni Cortes, Jordi Torres, Adrian Cristal, Osman Unsal, David Carrera, Yolanda Becerra, Enric Tejedor , Mateo Valero, "BSC vision on Big Data and extreme scale computing", BDEC Kukuoka white paper, 2014.

[2] Jesus Labarta, Eduard Ayguade, Rosa M. Badia, Yolanda Becerra, David Carrera, Toni Cortes, Enric Tejedor, Jordi Torres, and Mateo Valero, "Writing Efficient Computational Workflows in Python", BDEC Barcelona white paper, 2015.

[3] www.bsc.es/compss