

# *Data clustering and placement for the BaBar database*

## Abstract

The BaBar experiment at the PEP-II accelerator at SLAC, due to come online in early 1999, will generate approximately 300 TB of physics and simulated data per year during its lifetime. The user community of approximately 500 physicists is widely dispersed, but require high speed access to much of the data. The placement of data throughout the collaboration and across media of different access characteristics and costs will be crucial in allowing high speed, efficient access to the most frequently used information.

Placement or clustering strategies must perform in a fully multi-process and multi-user environment and must be able to utilize an arbitrary configuration of file servers (including hierarchical mass storage systems such as HPSS) in a distributed client-server architecture. Site-specific configuration must be able to cope with the requirements of the collaboration, regional and institutional centers.

BaBar is using object oriented database (ODBMS) technology as the basis for the event store and conditions database. This paper will describe the data placement strategy and its implementation in terms of clustering hints and configuration files.

The BaBar project is a High Energy Physics experiment at Stanford Linear Accelerator Center (SLAC). A central theme of its research program is the detailed study of the difference between matter and antimatter. The project is constructed by a large international collaboration of physicists and engineers from 10 countries. A new detector as well as the software are currently being designed and are due to come online in April 1999. Data taking is expected to last 10 years, leading to a total data sample of some 3 PB. Designing a system that can handle such enormous data volumes requires special tools and techniques.

One of important aspects of a system which is expected to handle petabytes of data is the issue of how to actually store data. Over half of BaBar data

will reside on tertiary storage (tapes), and careless data placement is simply unaffordable. Data should be placed in a way that will optimize its later access: “hot” data (like compressed event data, e.g. tag data) should go on fast media, while rarely used data should end up on tapes. Placement of data is undoubtedly one of the most important parts of the access optimization, the other equally important is efficient data clustering. Once a file is “touched”, as much data should be read out of it as possible; it is clearly undesirable to bring up a tape just to read several bytes.

In addition to basic requirements, which have to be fulfilled by the module responsible for placement and clustering, several others like

- full concurrency support
  - multiple file servers support
  - allow data distribution
  - allow site specific configuration
- must also be fulfilled.

The BaBar persistent data storage is based on two commercial products: an object database: Objectivity/DB and a mass storage system: HPSS. They introduce many new features and relieve the burden of writing low-level coding, but at the same time they add further restrictions and complications which should not be underestimated.

### *Data Structure*

The BaBar data has been structured into several independent pieces, with regards to its logical aspects. The outermost layer comprises of several basic “domains”:

- The Conditions database. This manages and tracks the conditions under which experimental data are acquired.
- The Event Store. This manages and tracks the experimental data from the initial raw data produced by the experiment or by simulations through the various reconstruction and physics analysis processing phases and selections.
- The Online Databases. These are databases that are specific to the online system. This part is not yet implemented.

Another two layers cope with the access rights to the contents of the database, disallowing unprivileged users to update the data. A database may belong to:

- System
- Group
- User

Every group has one entry under the “Groups” level, similarly every user has its private area under “Users”. For the sake of consistency there is one system user under the “System”. Authorization levels are domain-specific, and an application may execute at one authorization level for one domain and at another level within another domain.

The last level in the structure divides data into independent components, corresponding to:

- processing stage (in case of the Event Store):  
raw, rec, sim, aod, esd, tag
- subsystem (in case of the Conditions Database):  
pep, svt, dch, drc, emc, ifr, mag, tgr

Each BaBar Database<sup>1</sup> may belong to exactly one domain, one user<sup>2</sup>, and one component. In order to avoid locking problems between multiple processes writing the same type of data simultaneously, each process is assigned a separate container. That is not the case for the Conditions Database, where the process of updating the data is relatively short and occurs infrequently, thus clashes are unlikely to appear. Conditions Databases use another strategy, where more than one process is allowed to write to the same database. Should a clash occur, one of them will wait till the other “commit” and release locked container.

Should a process stop and leave a container partly filled, such a container be registered and reused later by other process.

### *Directory Organization*

Every layer described in the previous section corresponds to one directory level, only the last level contains files (=databases). In order to avoid having thousands of files we are splitting “component-level” into subdirectories containing fixed number of database files.

### *Multi-file System Support*

Because of the large size of stored data, we have implemented a mechanism, which allows us to

---

<sup>1</sup> A database in terms of the Objectivity/DB system is currently equivalent to a file. It consists of one or more “containers”, where container is a set of basic persistent object. Container is the locking granularity. In one of the next Objectivity releases each container (rather than a whole database) will be mapped to one file.

<sup>2</sup> “User” should be understand as a system, a group or a user.

utilize not only one file system, but virtually any number of file systems including HPSS. We require, that our daemon “pud” be running on each machine, where we store the data. The daemon is responsible for directory creation, available space verification and other file-system operations. A special configuration file provides full flexibility: each set of databases corresponding to a particular domain, authorization level, user, and component may be directed to different group of file systems. In addition, we have the influence on how the group of file systems is filled: it may be

- parallel (each new database is placed on the next file system in the group, or in the first one if the end of the list has been reached).
- sequential (databases are placed on the first file system till there is enough place, if the file system is full, the second one is used etc.)

The aforementioned configuration file allows us also to overwrite default quantities settings, like: size of a container, number of containers per database, or number of databases per one directory. Like groups of file systems, these parameters may be assigned independently to each domain, authorization level, user or component.

### *Database Naming Conventions*

In the Objectivity/DB environment, each database is assigned a name which has to be unique across the whole federation<sup>3</sup>. Since our collaboration is highly dispersed, it is common to have new databases

---

<sup>3</sup> A federation is a set of databases and its schema, the whole BaBar data will be stored in one federation.

created simultaneously at different places. In order to avoid name clashes<sup>4</sup> we decided to assign a pool of database numbers to each institute of the BaBar Collaboration, this allows us to assign to each database name truly unique number (uniqueness inside institute is handles by the software with no network communication). In addition to the unique number, each database name<sup>5</sup> contains of

- first letter of domain associated with it
- first letter of authorization level
- user name
- component name.

An example of a valid name:

*e\_u\_becla\_raw00345.bdb*

The current implementation does not contain unique database number allocation scheme, instead we use unique numbers inside each institute. [at the time of writing this document this feature was under tests]

### *Conclusions*

The described system has been successfully used for Mock Data Challenge II (MDC2), the primary goal of MDC2 was to simulate and reconstruct 500 000 events. We are confident, that the system which we have built (and are improving now) will be able to fulfill all requirements of the BaBar experiment.

---

<sup>4</sup> naming clashed might occur during data exchange, when databases from one institute will be shipped to another

<sup>5</sup> Excluding small number of “internal” databases.