OPERATIONAL FIELDS DATABASE MANAGEMENT

Dragan Jokic

European Centre for Medium-Range Weather Forecasts

Fields Database (FDB) is ECMWF's DBMS for the storage and retrieval of real time bulk data. It has been successfully used in the operational and research environment at ECMWF over the years. During the most recent years, the FDB has been re-designed, made a distributed database and improved to serve as a powerful database tool in both the super computer (MPP) environment and the local workstation environment. Structure, main features and functions of the FDB will be outlined as well as the FDB solutions for the massively parallel I/O. Also, database usage in the distributed operational and research environment will be discussed.

1. INTRODUCTION

Fields Database (FDB) is ECMWF's DBMS for the storage and retrieval of real time bulk data. It has been design and implemented in house. During most recent years FDB has been made into a distributed database, it supports massively parallel I/O and is object oriented.

There was a FORTRAN version of the database code until 1994. This version required full knowledge about database structure on the application level. Navigation through the database was not simple and required build-up of database records. Then, FDB was completely re-designed and written into c. Pilot Version 0.1 was installed on Cray YMP16-C90 and it also worked on Cray T3E. Migration of the whole of ECMWF's operations and research to the new database completed in 1995 when the new configuration and database structure were introduced. With the version 1.0 in 1996 FDB become a distributed database.After this, several configuration modes were added to the database, 'Multi-Server/Multi-Client' mode in 1997 - Version 2.0, 'MP I/O Multiple file systems' in 1997/1998 - Version 3.0. The latest version running in our operations is 3.3 which supports Parallel I/O in the workstation environment.

2. WHY DO WE NEED IT?

There are several reasons for ECMWF to develop/maintain its own database to output data from its analyses and forecast models and handle (retrieve/manage) on-line bulk date. The lack of commercial

support in this area is the primary reason. Several criteria such as efficiency, portability and flexibility must be fulfilled. On the other hand, the question is why database? It would be quite difficult and costly to care where data resides and how is it organised on the application level in the environment like ours. In our operations we operate on a tight time schedule. We use our data as soon as it is created, mainly to generate products for our Member States and as an input for other parts of our operational suites. Also, data is used for plotting and archiving as soon as created. Apart from these, we have other users using our data before it is archived. So, we need a common knowledge about the data and straight accessability throughout our systems. FDB makes data output-ed by our models immediately available and does not require any additional action on data to make it accessible.

Secondly, output generated by research experiments in terms of number of fields and volume of data is typically never the same. We need to support this flexibility in our system without any additional pre-settings. Also we want to make data available to our researchers for viewing, in the distributed environment, before data is archived.

Also, we need a system to which we can easily add new attributes without lots of changes and support new projects and testing of all different kinds.

Taking into account the current trend where users have more and more computer power at their disposal, we need something portable which would function the same way on the super computer and the office work station environment.

3. FEATURES

FDB supports variable length binaries in the database. It is of no importance what is the format of data. Could be really anything. There is a LOCKING mechanism and SIGNAL handler built into FDB to assure consistent status of the database when accessed by multiple users. Database administrator can restrict access to the database on any level and set up ownerships. Also, users can do this themselves when generating data by setting some FDB environment variables.

It is very easy to change database structure, add logical database or new attributes. FDB has Client-Server architecture. It supports massively parallel I/O. The latest releases can spread data over several file systems - 'MULTIPLE FILE SYSTEMS mode'. When writing data out from the parallel process no data positioning or offsets pre-setting is required. Parallel outputing is done in completely independent and asynchronous fashion.

It is easy to switch database I/O from one configuration to another just by using environment variables. FDB provides simple user interface which is the same for all configuration modes it can work in (standalone, distributed,....)

FDB has a feature to dynamically pre-allocate space prior to data write-out. It also has a feature not to write data directly to disk but into memory and then to empty buffers asynchronously in a multiple of disk sectors. It has several (env.) variables to control various buffers sizes and their configurations (all data/PE, for instance, for one forecast time step can be written out from the model in only one chunk which can be adjusted to disk sector sizes, etc).

Versions of the FDB software exist for the following platforms: FUJITSU VPP, SGI, SUN, HP and CRAY.

4. FDB ROUTINE OVERVIEW

When the new FDB was designed we had an idea to create a database which to users would simply look like a file. So there are several basic functions like open, attribute setting, read/write and close. Database is accessed by descriptor. No knowledge about database structure and attribute types and lengths is required on the application level. All attributes (in our case MARS parameters alike) which characterise data are defined by database 'logical name' (fully defined in the 'Database dictionary'). The following is the list of the basic FDB functions:

Initialisation Routine	*initfdb();*
Open Database	*openfdb(logical_name,desc,mode,stat);*
Manipulate Attributes	*setvalfdb(desc,attribute_name,attribute_value);*
Manipulate Records	*writefdb(desc,data,len); readfdb(desc,data,len);*
Close Database	*closefdb(desc);*

Both c and FORTRAN interface are available to users. There are more functions then these above and they are mainly used by the applications which access both local and remote databases simultaneously.

Database configuration and all attributes which characterise databases are defined in 'Database dictionary'. 'Database dictionary' is a simple ASCII file containing the logical database names, attribute names and some elements defining database structure. The outlook of the file, is too technical and will not be discussed here. By simply modifying data dictionary, logical databases can be created, new attributes added or existing database structure changed. No changes are required on the source code level.

5. FDB CONFIGURATIONS

FDB can work in several configuration modes:

- Standalone;
- Client-Server;
- Multi-Client Multi-Server;
- IFS (MPP);
- Multiple File Systems.

5.1 Standalone mode

Standalone configuration mode has been designed for a single processing unit which has disks connected to it. It has dynamic indexing and data is not written directly onto disk, but into memory. Buffer(s) are emptied in multiplies of disk sectors. It supports more then one file system.

The advantage of this configuration is that the full control over database is done from within a single process. Data is centralised and archiving can be done in a simple meaner. Standalone configuration mode is shown in Figure 1.

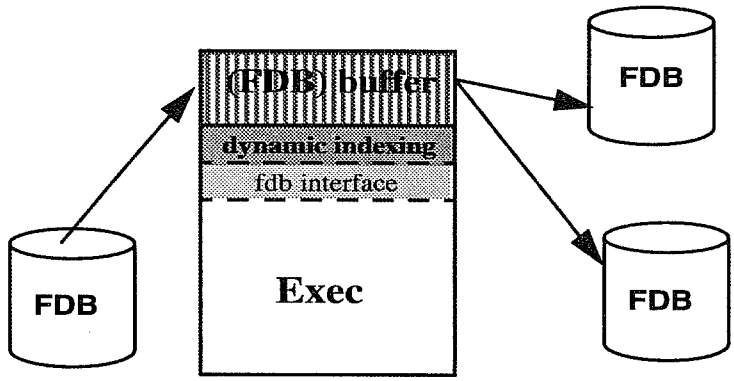

Figure 1: Standalone Configuration Mode

In ECMWF's operations, Figure 2, FDB is used in this configuration within Products Generation, which is a parallel application where each processing element accesses the database independently in standalone mode. Also, data is retrieved on the data home host by MARS using this configuration mode.

Operational o suite;
Fujitsu VPP700

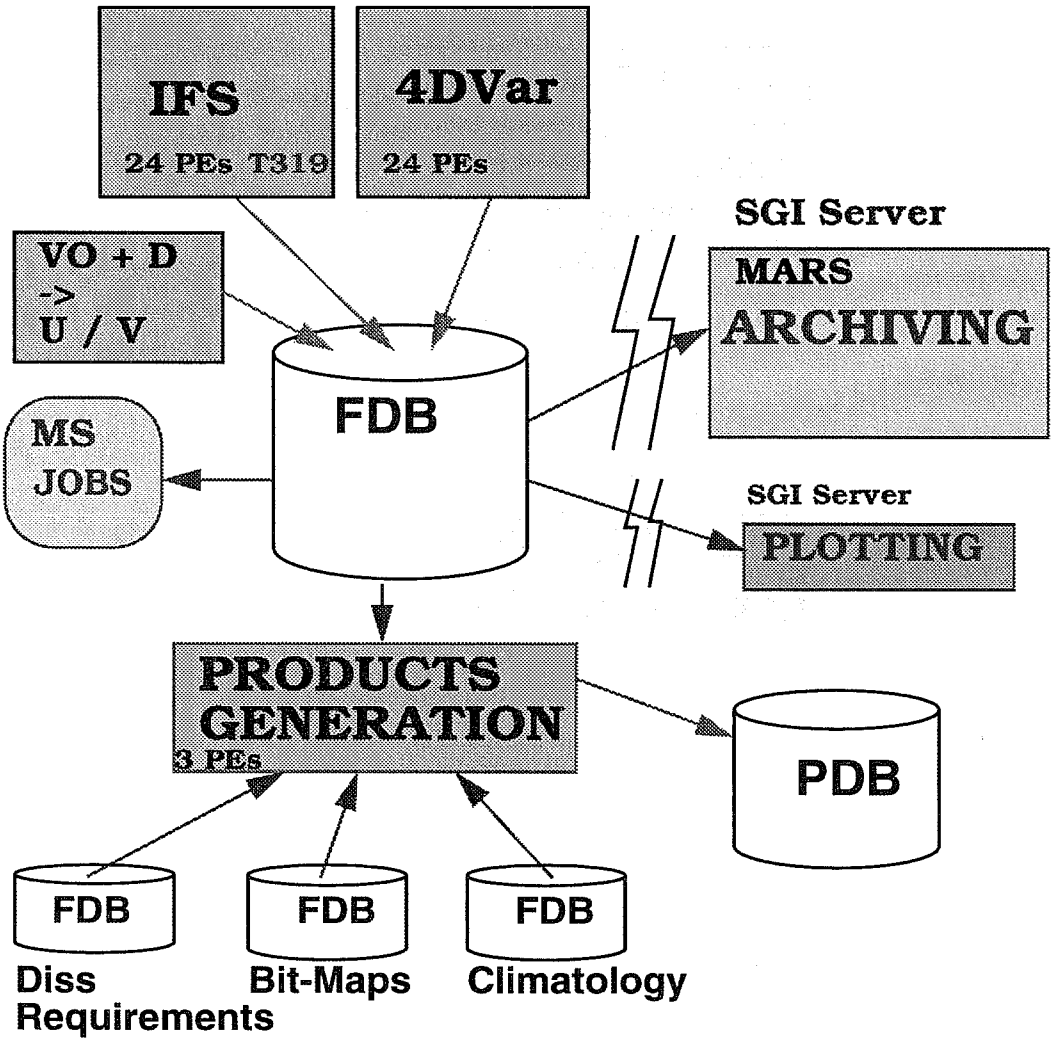

Figure 2: FDB usage in the operational 'o' suite.

5.2 Client-Server configuration mode

Client-Server configuration consists of two processes typically running on two different hosts. Application process is the client process. Client process does not have to have any knowledge about how data is organised on the server machine. Could be different from the local one. With the FDB Version 2.0, buffering was introduced on the client side so distributed applications can write data remotely in an asynchronous fashion.

rpcgen is used as the protocol compiler and XDR functions for the transferring of arbitrary structured data packets. Client-Server configuration is shown on Figure 3. FDB servers are run on every IMPE of our Fujitsu machines, on some SGI servers and HP.

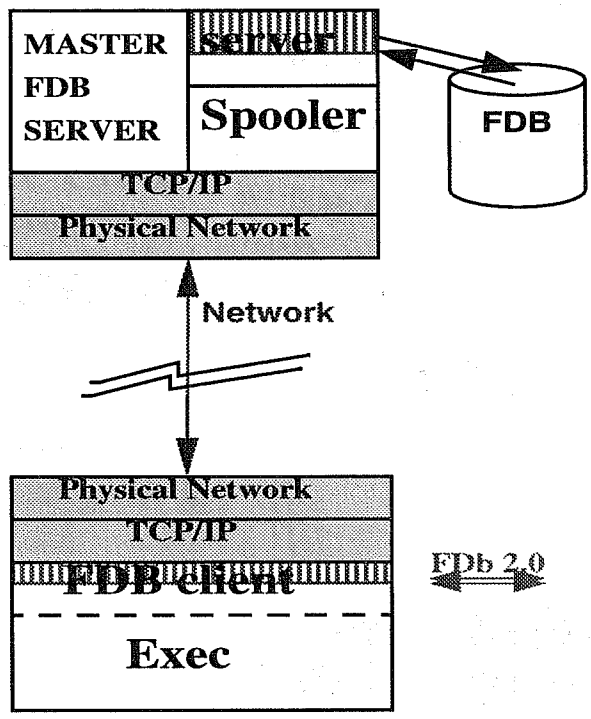

Figure 3: Client-Server Configuration Mode

FDB is used in Client-Server configuration for several purposes. One is to retrieve operational and research data for archiving (Figure 2/Figure 4). Data is typically generated on one of our Fujitsu machines and the archiving is run on one of SGI servers. Also, FDB is used in this configuration for plotting which is run on SGI servers and for the retrieval of on line data for viewing using ECMWF's METVIEW.

5.3 Multi Client configuration mode

The idea with this configuration is to have an FDB server which is capable of serving requests from more then one client, actually, tens of clients simultaneously. Client processes in this configuration are typically processing elements of a large parallel process. When a number of large parallel processes are run each process would be separately served by a so called FDB 'select' server.

The advantage of this configuration is that full control over database is done from within a single process. Server can run on any machine so parallel applications can write data out remotely. Multi-Client configuration is shown in Figure 5.

FDB is currently used in this configuration to write data out from processes which run in parallel, access database simultaneously and output lots of fields like clusters / probabilities / tubes jobs shown in Figure 4.

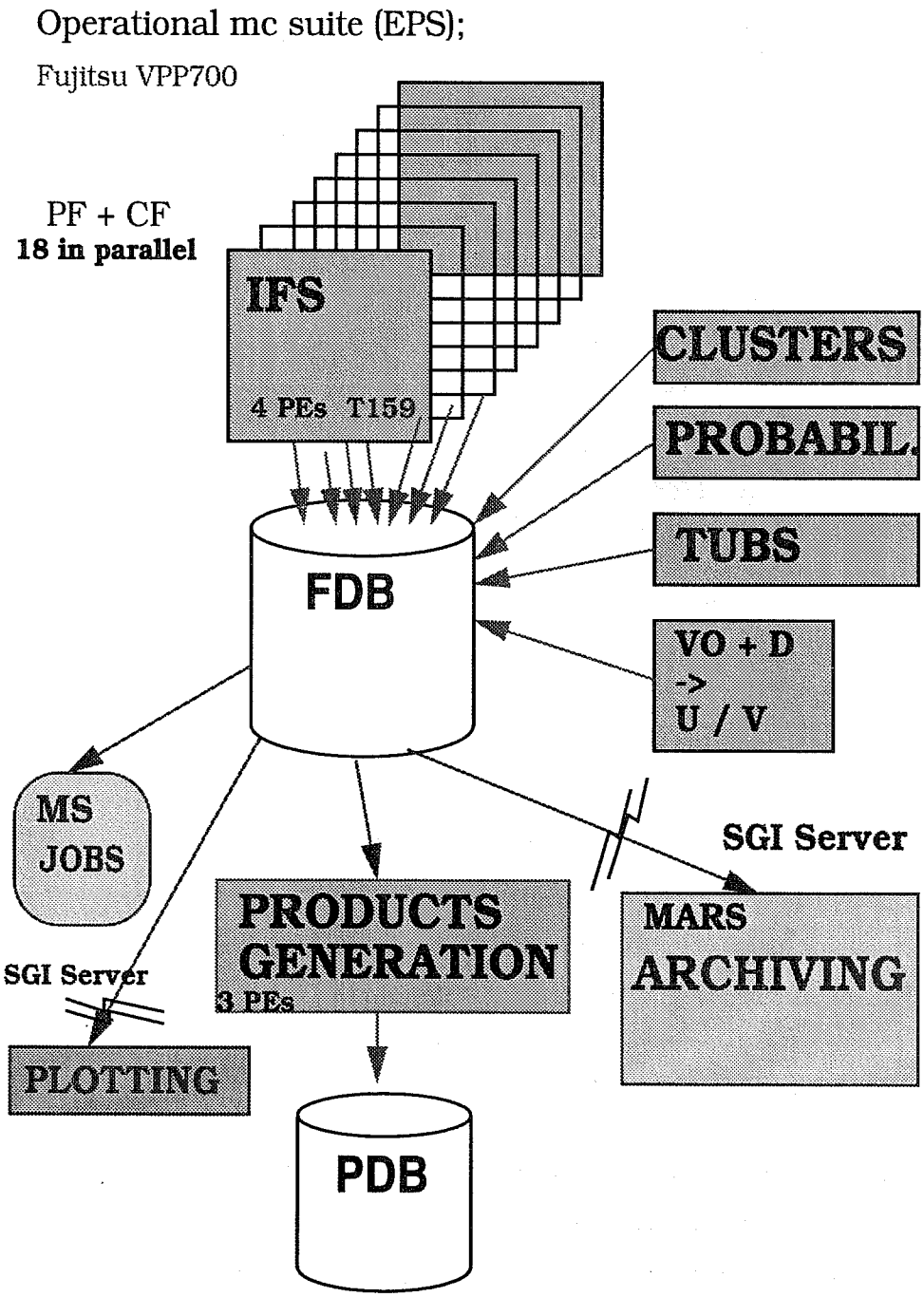

Figure 4: FDB usage in the operational 'mc' suite.

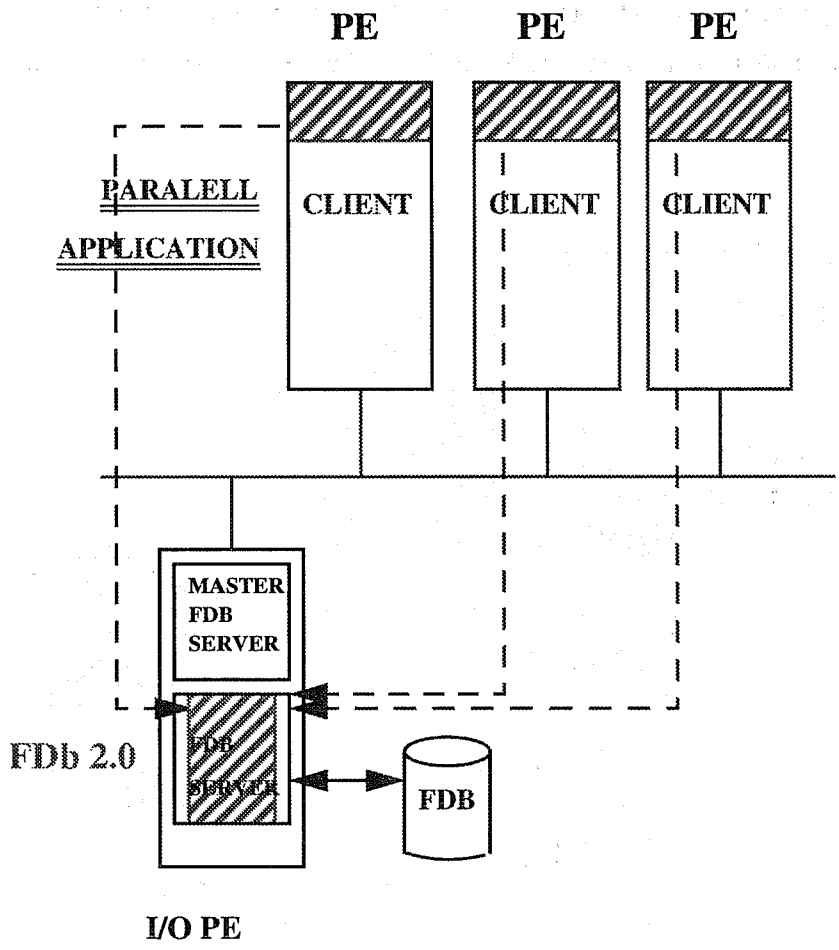

Figure 5: Multi-Client Configuration Mode

FDB was originally used in Multi-Client configuration mode when our first Fujitsu vpp machine came in. All tests performed well. In reality, large numbers of research experiments created a heavy TCP/IP load on the machine which affected performance.When buffering was introduced on the client side, with FDB Version 2.0, it all went back to normal.

5.4 IFS (MPP) configuration mode

When asynchronous I/O become available on Fujitsu we introduced so-called IFS configuration mode. In this configuration mode each processing element of a large parallel process writes data out independently of each other. There are also some buffering techniques involved. Meta data are on the other hand written and kept together which is the key to making the database consistent and data immediately accessible. Although the data is written into separate files, the database is visible to users as a single virtual entity.

Bulk data are kept on so called VFL file systems. The space is dynamically pre allocated prior to writing. Meta data is kept on so called UNIX file system. IFS configuration is shown in Figure 6.

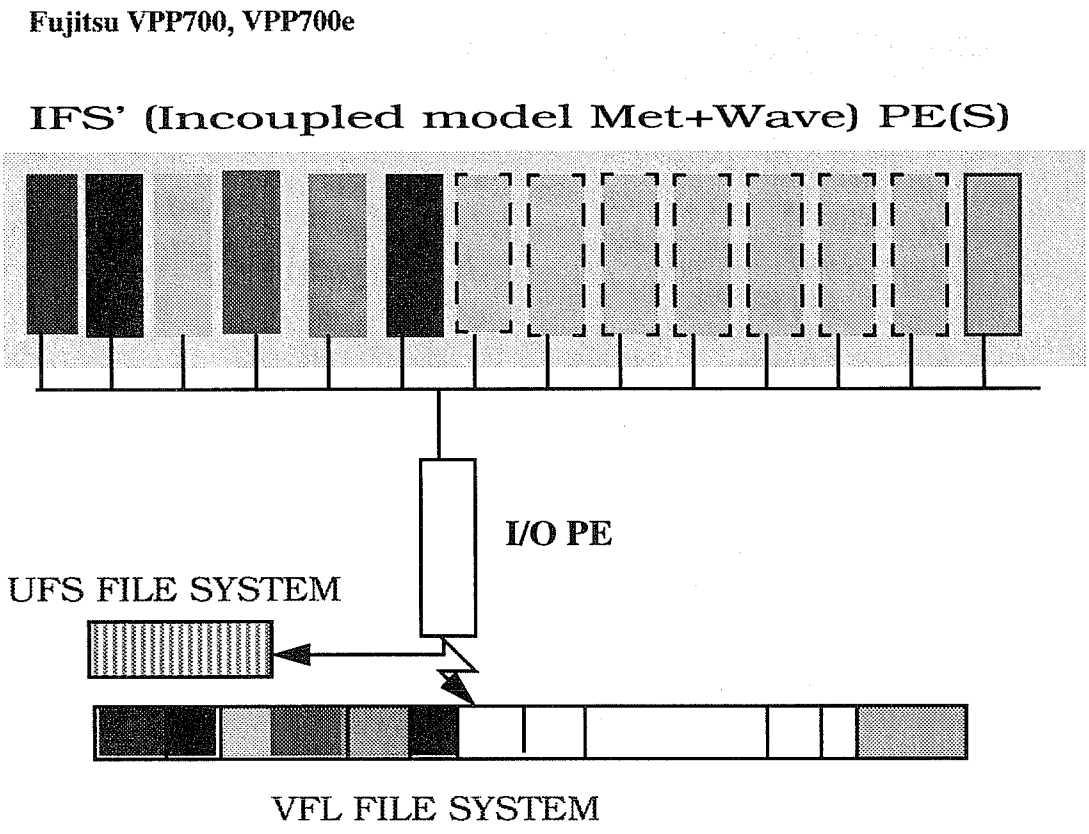

Figure 6: IFS Configuration Mode

FDB is used in this configuration in both operational and research environment to output data from IFS and 4DVar analyses. In operations (Figure 2), main IFS (ten day forecast) is at the moment run on 24 PEs. Each operational 4DVar analyses is also at the moment run on 24 PEs. In research, IFS/4DVar can run on any number of PEs depending on the model resolution and system loads. In the operational 'mc' suite, we run 50 perturbed forecasts and a control forecast. 18 of these are run in parallel, each on 4 PEs, all simultaneously feeding the same database.

5.5 Multiple File Systems

The latest FDB configuration can spread data over several file systems. It is quite complex to know where data actually resides and to make IFS (or any application) re-runnable, but we found a solution and manage to do it. In this configuration I/O is spread over several I/O PEs (controllers). The number

of file systems can be set simply by the environment variable. As in previous configurations, the database looks to users as a single virtual entity. Figure 7 shows Multiple File Systems configuration.

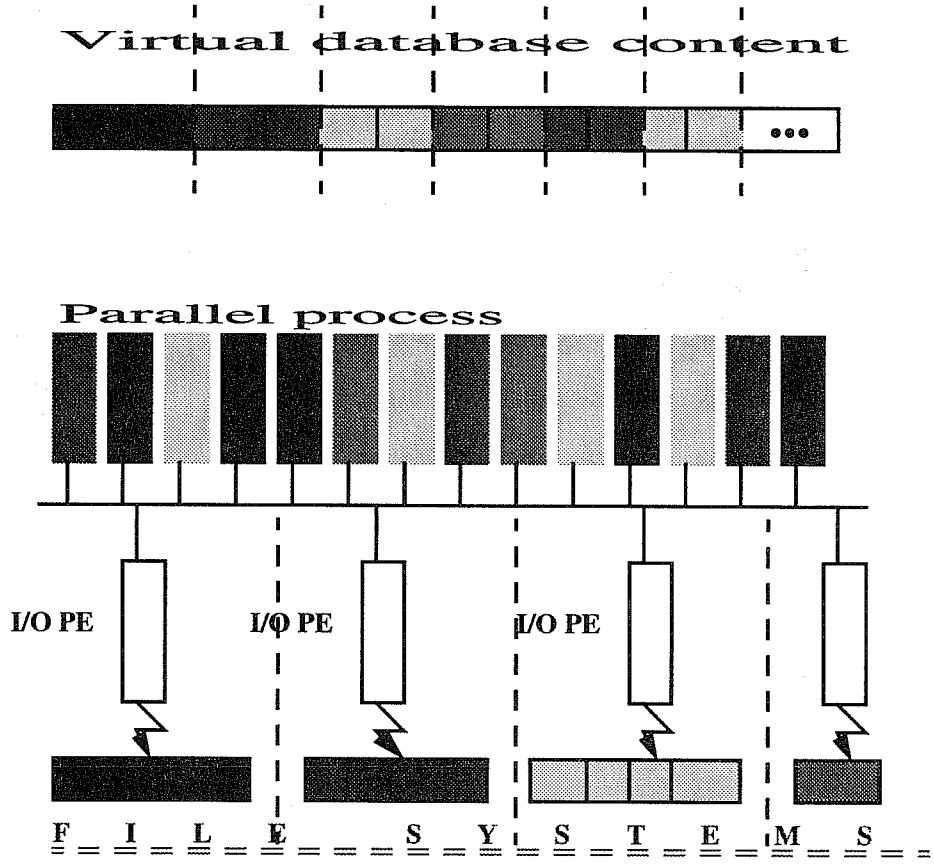

Figure 7: Multiple File Systems configuration mode.

6. PERFORMANCE

Performance of the FDB is independent of the number of fields and the volume of data in the database.

FDB performs dynamic indexing during production process and performs automatic data partitioning.

Figure 8 is a performance graph of the latest operational IFS (T319L60). Number of PEs on which the model is run is shown on the x axis. Test is run from 16 to 98 PEs. One can see how well FDB curve is following IFS for a different number of processing elements (PEs). Closeness of curves is due to asynchronous nature of the FDB. At the moment operational IFS is ran on 24 PEs on Fujitsu vpp700.

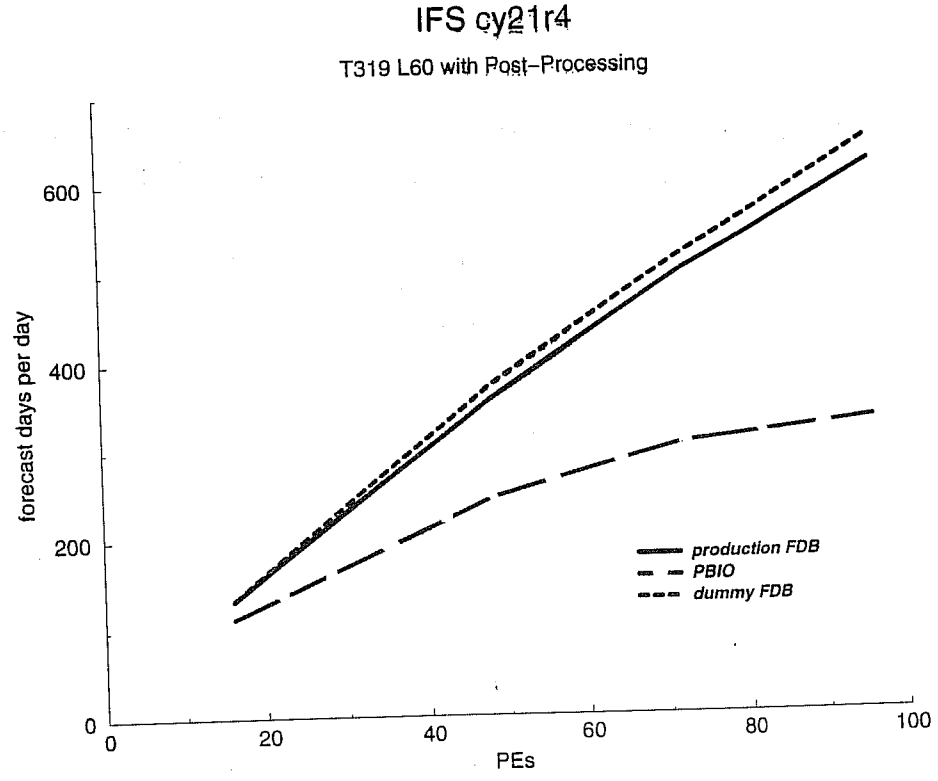

Figure 8 IFS cy21r4

7. FUTURE

Next platform on which FDB software will be installed is Fujitsu VPP5000. Up until now ECMWF has had to rely on its own tool to output data efficiently. In order to achieve top class performance FDB has had to follow machine architecture. Whether some general tool for efficient I/O on next generation computers is going to be available is yet to be seen.