Computer Aided Design Facilities for Prototyping the Omega DBMS

Mikhail L. Zymbler
Chelyabinsk State University
Chelyabinsk, Russia
mzym@csu.ac.ru

Abstract[1]

The paper describes a set of computer aided design facilities, used for prototyping the parallel DBMS (Database Management System), called Omega. This system is designed for a MVS-100/1000 massively parallel computer system. These computer aided facilities include both software tools from third-party vendors and those especially designed for this project. The former are a UNIX/Linux operating system, C++ compilers for MVS-100/1000 and IBM PC, CVS (a Concurrent Versions System), DOC++ (a documentation system for C/C++), Router – a distributed operating system for MVS-100/1000, etc. The latter consists of an operating environment, a profiler and a debugger. The paper proposes a certain way of integrating these software tools for increasing the productivity of the cooperative development of such a complicated software as a parallel DBMS for a supercomputer.

1. Introduction

The effective development of complicated software systems is currently impossible without modern software engineering. According to the definition given in [1] *Software Engineering* is a set of conceptual, management and programming tools used in software development. This paper touches upon the problem of choosing software facilities for the Omega parallel Database Management System (DBMS) [2] for a MVS-100/1000 multiprocessor computer system [3].

A developer's tools can be formed in two ways. The former consists in using an integrated development environment (the so-called *technological complex*) to support all the basic stages of the software development cycle. A great number of such technological complexes have already been implemented (see, for instance [4-7]). The basic disadvantage of this approach is developers' strict dependence on a specific software which limits their abilities. To use some function absent in the given technological complex you either should improve that complex or chose a new complex. Both situations lead to extra expenses not connected with the program system under development. Using a specific technological complex also limits the developers' choice of a hardware platform. Such an approach turns out to be productive in the development of applied software for specific problems, take for instance, developing applications for database processing [8].

The latter approach is valid for selecting a set of tools, each of them supporting some stage of the development cycle. The basic advantage of this approach is high flexibility in forming the developers' technological environment. Even within a single project, they can select specific components of a technological environment according to their individual aptitudes. In this case it's relatively simple to replace some component which ceases to meet the project's or the developers' requirements. The main problem of such an approach is the correct choice and integration of the project technological environment. This approach is most frequently usable in the development of system software.

We'll show in this paper the structure of computer aided design facilities used in the development of a parallel DBMS prototype for a MVS-100/1000 system. Section 2 gives a brief description of the MVS-100/1000 architecture and the Router distributed operating system. Section 3 describes the Development Environment of Omega DBMS including both basic software tools from third-party vendors and those especially designed for this project. Section 4 gives the description of project participants' workplace and the technological cycle of the development process. The

[1] This paper has been supported by the Russian Foundation for Basic Research (grant No. 97-07-90148).

Proceedings of the Workshop on Computer Science and Information Technologies CSIT'99
Moscow, Russia, 1999

conclusion of the paper carries an analysis of the efficiency of the tools used and a possibility of embedding such tools into similar projects.

2. MVS-100/1000 Architecture

MVS-100/1000 supercomputer was designed jointly by Science Research Institute "Kvant", Institute of Applied Mathematics by Keldysh of Russian Science Academy, Science Research Institute of Theoretical Physics and Institute of Mathematics and Mechanics of Ural Branch of Russian Science Academy. According to the classification given in [9], MVS-100/1000 corresponds to a multiprocessor system with the cluster architecture. In conformity with Flynn taxonomy [10] MVS-100/1000 architecture can be considered as MIMD.

The MVS-100/1000 consists of *processor modules*. Each module is a double-processor computer having *computation* and *communication processors*. The MVS-100/1000 uses the superscalar Intel i860 processor as computation processor. The communication processor addresses all RAM of a processor module, the computation processor addresses only some part of RAM. Synchronizing of computation and communication processors is implemented on the hardware level. The only external devices of a processor module are speed channels, namely *links* of the communication processor. The communication processor has about four (MVS-100) or six (MVS-1000) links. By means of such links processor modules are connected with each other and a disk subsystem. The MVS can include hundreds of modules and several disk subsystems organized as a module network.

The network obtained in such a way is connected with the control computer (*host-computer*) via one link. IBM PC compatible workstation is used as a host-computer.

OS Router designed by the Institute of Applied Mathematics by Keldysh of the Russian Science Academy is used as a MVS-100/1000 operating system. The host-computer usually runs under UNIX or Linux.

OS Router is a distributed operating system of the toolset class. It consists of a processor server for running on each communication processor and the host-server (iserver.exe) for running on the host-computer. Both processor server and host-server are always loaded before the user's task.

OS Router provides the following basic functions:

- loading the user's programs from the host-computer to processor modules;
- interchanging data between the user program and the host-computer in the form of some subset of UNIX input/output functions;
- interchanging via links among programs run on different processors.

The user's task can be run only on computation processors. This task is a totality of processes. Each processor module can execute the only one process of the user. Each process is implemented as program, that is, as individual load module of a programming system. Program text has calls of OS Router functions, which realize a message exchange among other programs. There are no tools to provide a program development for the whole processor network.

3. Omega Project Development Environment

The Omega project Development Environment (DE) consists of software tools from third-party vendors and those especially designed for this project. The former set includes GNU Emacs text editor, CVS versions control system, DOC++ (all under Linux), Portland Group C++ compiler (PGCC), Borland C++ compiler for Windows and some UNIX utilities. The latter set consists of Debugger and Profiler (all for i860).

The main purpose of forming the Omega DE is to create suitable environment supporting the basic stages of developing lifecycle of C system software for the MVS-100/1000. Such an DE is supposed to guarantee the project participants' effective work.

3.1 Software Tools from Third-party Vendors

The basic software tools from third-party vendors used in the Omega project are CVS versions control system and DOC++ source code documentation system.

CVS Versions Control System

CVS (concurrent versions system) [11] is the basic tool to provide the Omega project versioning and to support the developers' co-work.

CVS runs under UNIX and can be used by a developer as CASE-package (toolkit) on the stages of coding and maintenance. CVS supports the concepts of the *repository* and the *working copy* of the project. The repository stores a complete copy of all the files and directories which are under version control. No developer ever accesses any of the files into the repository directly. Instead, he uses CVS commands to get his own copy of the files and then work on that copy.

Having made a set of changes, each developer is to commit (check-in) them back into the repository to make them accessible for other developers. Each developer has to update his working copy periodically to get changes from other developers. While checking-in the developer is to comment on his changes for the repository to contain all the changes whenever and whoever made by. During updating and checking-in CVS automatically merges the changes from other developers, if these changes do not affect the same lines of the source code. If modifications overlap the resulting file includes both versions of the overlapping lines, which are delimited by special markers. The developer is to resolve this conflict manually to finish the checking-in.

The developer can get the history of changes of the specified file and compare two versions of this file. He can also get its current status, which is up-to-date, locally modified, needing a merging, having an unresolved conflict etc.

Each version of a file has a unique revision number which changes automatically after checking-in. Revision numbers look like `1.1', `1.2', `1.3', or `1.3.2.2' and even `1.3.2.2.4.5' in case of the project's branch. Project files (called release) with specified revisions can be assigned to the same symbolic tag. For instance, a release may have name `release-1', `release-2', `release-1_patches'. The developer can check-out the specified file or the whole release using their tags.

While debugging or maintaining previous project releases there often arises a necessity to change something in them and then to spread modifications to the current release. CVS supports the concepts of the main trunk and branches of the revision tree. To solve the problem the developer can check-out a necessary release, then create a branch and make modifications in this branch (without affecting the main trunk). When the modifications are made the developer can select either to incorporate them into the main trunk and check-in, or to leave them in the branch.

CVS seems to be very useful for cooperative development of some complicated software systems.

DOC++ Source Code Documentation System

DOC++ source code documentation system [12] helps to create high quality documentation on a program system immediately while the editing of C/C++ sources. Such an approach allows to fulfil the code self-explicability requirement as well as to avoid the duplication of the information in documentation and source codes. This is the most effective way to secure the adequacy of documentation and its conformity to source codes.

The main purpose of documentation system is to include into the source text comments of a special format:

/**...*/

Such comments, unlike those with only one preceding asterisk will be analyzed by DOC++ and converted into HTML format. At the same time documenting system uses the number of reserved words started with `@' and placed into a special comment. The following specification of a function is to be processed by DOC++:

```
/**
Create a new thread and calculate its T-
factor. There is no transmission of control
to the new thread. In fact th_fork() creates
a new record in the threads table of the
thread manager. If the factor-function is
omitted it is assumed to be
(TH_FACTOR_MAX/2). If priority is
TH_LOWEST_NICE then the given thread will be
run only in the absence of other ready-to-run
threads with higher priorities.
@memo Create a new thread
@param proc - pointer to function
representing thread body
@param param - pointer to parameter list or
NULL
@param factorfn - pointer to factor-function
or NULL
@param type - a thread type:
\begin{verbatim}
       TH_AND - conjunction
       TH_OR - disjunction
       TH_SYS - system
\end{verbatim}
@param nice - thread priority (integer number
from -20 to 20).
@return - the positive tid of a new thread or
a negative error code:
\begin{verbatim}
TH_OVERFLOW - an overflow of the threads
table,
TH_ENOMEM - not enough memory to create a new
thread
\end{verbatim}
@see th_proc_t
*/
extern th_tid_t th_fork(th_proc_t proc, void*
param, th_factorfn_t factorfn, th_type_t
type, th_nice_t nice);
```

The source codes like those given above will be automatically compiled into high quality HTML documents.

The main advantages of such an approach are a) permanent adequacy of documentation b) possibility to use documentation under any hard/soft-ware platform with HTML browser c) possibility to use documentation via Internet.

3.4 Software Tools Especially Designed for the Omega Project

Debugger

PGCC for MVS-100/1000 Development Environment has no internal debugger. Therefore within the frame of the Omega project a *Debugger* has been designed.

The Omega Debugger provides developers with unified tools for program debugging. All the functions of the Debugger are embedded into the object code if the OMEGA_DEBUG special pre-processor symbol is defined. The Debugger has the following basic functions:

- `void` **`db_setMode`**`(char mode)` - set the debug mode;
- `void` **`db_setScale`**`(long bitscale)` - set the processors debug print scale;
- `void` **`db_printf`**`(char * pointid, char * fmtstr, …)` - show the values of variables.

The `db_setMode(mode)` function call secures the setting either a step-by-step mode or a dump mode for the `db_printf()` function. In the step-by-step mode each debug message requires press Enter to continue. In the dump mode the debug message has no pause, it is assumed that the developer redirects output to a file.

The `db_setScale(bitscale)` function call allows to specify CPU numbers for which debug messages will be shown. If the i-th right bit of argument is set then debug

messages are printed for the respective i-th processor. All the bits of the debug print scale are set by default.

The `db_printf(pointid, fmtstr, …)` function call outputs debug messages of the following unified format:

`<call>: <CPU>: <point id>: <formatted argument list>`,

where

- `call` is the serial number of `db_printf()` call;
- `CPU` is the current processor's number;
- `point id` is a string to identify the place of `db_printf()` call in the program. Developers are recommended to use the following format of `point id`:
  <the name of the function >: <the serial number of db_printf() call in the function>
- `fmtstr` is a string to format an argument list.

Such a debugger, of course, can not replace a full-value symbolic debugger. Still it helps to increase debugging effectiveness.

Profiler

One of the most important problems to arise in the DBMS design and its realization is query optimization. The most time valuable things during the query execution in parallel DBMS for cluster architecture platforms are the data passing among processor nodes and transfers among disks. Therefore to estimate any query execution plan one must know the amount of all the data which are involved in the interchanges among disks and transfers via links.

So within the frame of the Omega project a *Profiler* has been designed to solve such problems for the MVS-100/1000 multiprocessor system. All the functions of the Profiler are imbedded into the object code if the OMEGA_PROFILE special pre-processor symbol is defined. The Profiler has the following basic functions:

- `int` **`pf_createTag`**`(int tag)` - create a tag;
- `int` **`pf_startProfile`**`(int tag)` - start profiling by a specified tag;
- `int` **`pf_stopProfile`**`(int tag)` - stop profiling by a specified tag;
- `long` **`pf_getValue`**`(int tag, int type)` - get a value of a specified type for a specified tag;
- `int` **`pf_addValue`**`(int tag, int type, long value)` - update a value of a specified type for a specified tag.

The `pf_createTag(tag)` function call creates a record with a specified tag in the profiler table. This record has fields to accumulate interchanges among disks and transfers via links initiated by the given thread. In fact a tag is some kind of identification which allow users to sum profile results received after the execution of some query on several processors.

The `pf_startProfile(tag)` function call switches on a profile mode for the specified tag. This call does not clear interchange and message counters i.e. new values will be added to the old ones.

The `pf_stopProfile(tag)` function call switches off a profile mode for the specified tag. This call does not clear interchange and message counters i.e. a developer can continue profiling for the specified tag later.

The `pf_getValue(tag, type)` function call returns the value of the specified type for the specified tag. Here `type` indicates a type of the value: is it interchange among disks or transfer via links.

To profile disk interchanges and transfers via links of some thread developers are to use the `pf_addValue(tag, type, value)` function call to update the respective (in the sense of the `type` argument) counters of all the profile records with a profile mode switched on.

4. Omega Project Cooperative Development Organization

4.1 Network Environment

The designed Omega DE proposes two different variants of the project participants' workplace (see Figure 1).

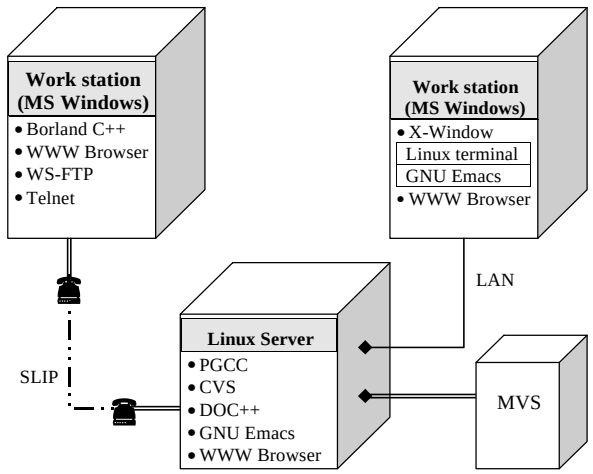

Figure 1. Network Environment Structure of the Omega DBMS Development

The former provides Borland C++ Integrated DE (IDE). The sources are stored on developer's workstation. After editing, the developer transfers those files to Linux-server via `ftp`. In this case `telnet` is used as a Unix-terminal. `telnet` also helps to run PGCC compiler on the Linux-server, after that it runs a task execution on the MVS-100/1000. Both variants use the standard UNIX `make` utility.

The latter variant is fully oriented upon Linux. GNU Emacs is used as a text editor because it has built-in C and CVS support. In fact, Emacs plays the role of some kind of shell which provides C sources editing, compiling and running. The developers have workstations with Windows 95/NT installed. The workstations are connected into a local network with Linux-server as the host-computer of the MVS-100/1000. Developers use X-terminals to deal with the Linux-server.

The first variant has two important advantages. Firstly, it allows working via Internet which would be impossible in the first variant with the inadmissible slow speed of X-terminal for data transferring. Secondly, Borland C++ IDE can be used for compiling, running and, what is most important, debugging hardware-independent parts of the sources because Borland C++ IDE unlike PGCC has a powerful built-in Debugger.

4.2 Technological Cycle of Cooperative Development

Working on his module of the project each developer uses directly only Profiler and Debugger of the project DE. Project versioning (connected with CVS) and source code documentation support (connected with DOC++) is under responsibility of the Librarian Programmer of the project (see Figure 2). He is also in charge of making the project and all-system testing.

A Developer can not check-in changes in his working copy into the project repository directly (CVS provides such a possibility). He can only check-out or update his working copy using special scripts (precisely check-out or update parts of project the corresponding to his module). At the same time the Developer can use CVS to support his repository for his own purposes.

A Developer must comment upon his sources correctly in the sense of DOC++. Moreover, a Developer must design program(s) for his module testing. After that he sends the module sources and the program for testing to the Librarian Programmer via email.

Here is a sample technological cycle of development:

a) a Developer sends modified source and change log to the Librarian Programmer;

b) the Librarian Programmer makes Developer's module and executes a module test, then makes an all-system test;

c) in case of any error a Developer will get a test log and a notification about the necessity of further correction;

d) in case of success the Librarian Programmer checks-in the modified sources commenting upon changes by the log (from a) step);

e) the Librarian Programmer generates a new version of the project documentation, if needed;

f) the Librarian Programmer notifies the Project Manager and then the Project Manager notifies the rest of the developers.

Project Manager is the only person in charge of making decisions to tag the project's release or to create a project branch.

5. Conclusion

The presented technological cycle and its software facilities could be used in the cooperative development of complicated software systems for various purposes. The Development Environment proposed here allows taking into account individual aptitudes of the project participants. Such an environment guarantees effective work under any hard/soft-ware platform either in a local network or via Internet.

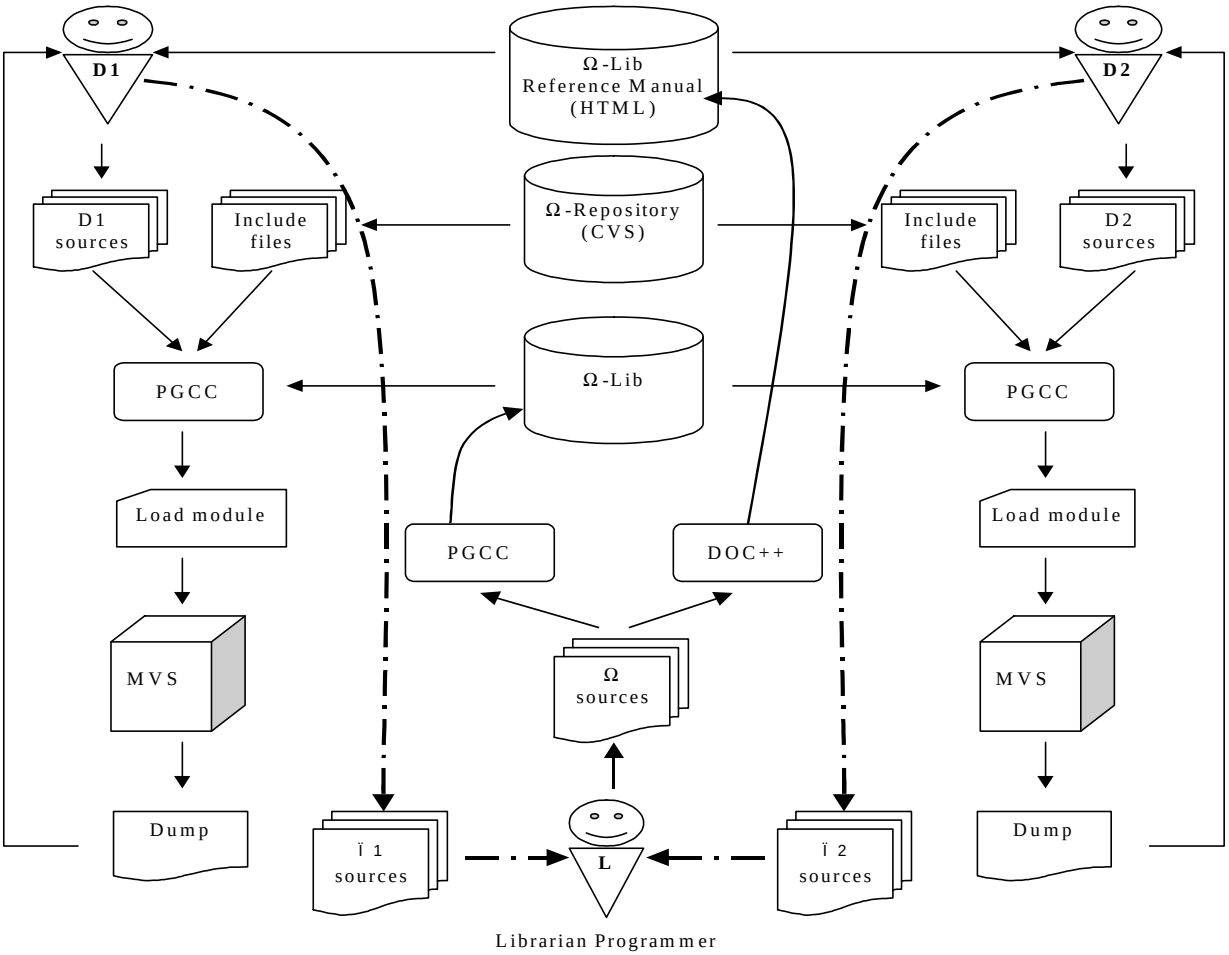

Figure 2. Cooperative Development Organization for the Omega Project

References

1. Safonov V.O. "Programming Languages and Methods for Elbrus Computer System". Nauka, Moscow, 1989 (in russian)
2. Sokolinsky L., Axenov O., Gutova S. "Omega: The Highly Parallel Database System Project". In: *Proceedings of the First East-European Symposium on Advances in Database and Information Systems (ADBIS'97)*, St.-Petersburg. September 2-5, 1997, vol. 2, pp 88-90.
3. Zabrodin A.V., Levin V.K., Korneev V.V. "The Massively Parallel Computer System MBC-100". In: *Proceedings of PaCT-95*, 1995, pp 342-356 *(Lecture Notes in Computer Science*, vol. 964)
4. Velbitsky I.V. "Software Engineering". Kiev, Tekhnika, 1984 (in russian)
5. Melnikov I.A., Raabe A.S., Tamm B.G. "Computer Aided Facilities Supporting Software Lifecycle. Review of Foreign Tools". *Applied Informatic* 1988; 14: 16-40 (in russian)
6. Sokolinsky L.B. "Programming Support for TIP-technology on Elbrus Multiprocessor System". In: *Proceedings of the First Regional Conference on Software Engineering, Applied and System Software*. Perm: PermSU-press. 1989. pp. 32-33 (in russian)
7. Fuksman A.A.. "Theoretical Aspects of Programming Systems Development". Statistika, Moscow, 1979 (in russian)
8. Pozin B.A. "Modern Software Engineering Tools for Development Open Applied Information Systems" *DBMS* 1995; 1: 139-144 (in russian)
9. Pfister G. "Sizing Up Parallel Architectures". *DataBase Programming & Design OnLine* 1998; May (http://www.dbpd.com/9805feat.html)
10. Flynn M.J., Rudd K.W. "Parallel architectures". *ACM Computing Surveys*. 1996; March. Vol. 28. No. 1: 67-70
11. Berliner B. "CVS II: Parallelizing Software Development" http://www.hu.freebsd.org/hu/doc/psd/28.cvs/paper.html
12. Wunderling R., Zöckler M. "DOC++. A Documentation System for C/C++ and Java". http://www.zib.de/Visual/software/doc++