DISTRIBUTED MEMORY IN A HETEROGENEOUS NETWORK, AS USED IN THE CERN. PS-COMPLEX TIMING SYSTEM

V. Kovaltsov[1], J. Lewis
PS Division, CERN, CH-1211 Geneva 23, Switzerland

ABSTRACT

The Distributed Table Manager (DTM) is a fast and efficient utility for distributing named binary data structures called *Tables*, of arbitrary size and structure, around a heterogeneous network of computers to a set of registered clients. The Tables are transmitted over a UDP network between DTM servers in *network format*, where the servers perform the conversions to and from *host format* for local clients. The servers provide clients with synchronization mechanisms, a choice of network data flows, and table options such as *keeping table disc copies, shared memory or heap memory table allocation, table read/write permissions*, and *table subnet broadcasting.* DTM has been designed to be easily maintainable and to recover automatically from the type of errors typically encountered in a large control system network.

The DTM system is based on a three-level server daemon hierarchy, in which an inter daemon protocol handles network failures and incorporates recovery procedures which will guarantee table consistency when communications are re-established. These protocols are implemented over a communications layer which performs the data conversions, packet splitting, error-correction with retry and timeout handling. The same communications layer is used to implement the application program interface which calls on the server daemon for client services. DTM is a registration based system, in which communications are established dynamically as needed and tables are distributed only to the clients who have registered their interest in them. The registration protocols include mechanisms to recover from daemon re-launches and to clean up after aborted clients.

1. INTRODUCTION

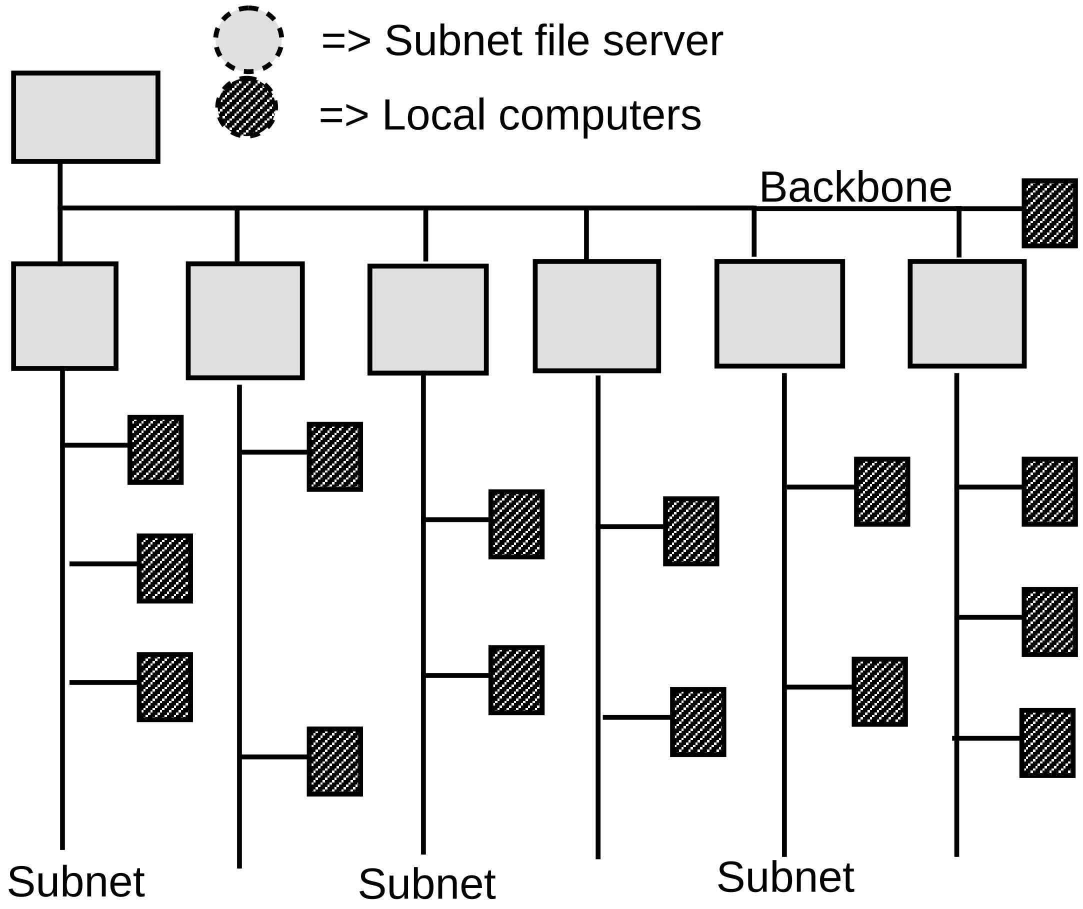
The PS Complex contains nine accelerators which interact in a time-sliced manner to produce particle beams varying in particle type, energy and time structure. The timing and sequencing of the accelerators in the network is controlled through a Master Timing Generator (MTG) which distributes events over a specialized timing network to the nine accelerator control subsystems [1]; some of these events are also distributed over the computer network. DTM complements the standard RPC-based PS equipment access, by providing *notify-on-change* events and *data-transfer* mechanisms, which are needed in any large control network and in particular by timing and sequencing mechanisms.

The *PS-Controls* general purpose computer network [Fig. 1], contains an isolated subnet for each accelerator and one main control system backbone which inter-connects them [2]. Each subnet contains its own file server, workstations and diskless VME-based front end processors and each must be able to function independently with its link to the main backbone cut. This network topology and subnet isolation requirement implies multiple copies of the common data needed to run each subnet independently and hence a mechanism capable of tolerating network failures, which correctly distributes pending data modifications when the fault disappears.

The implementation chosen for the timing system and its application program interface, implies almost instantaneous access to a number of common volatile data structures which are distributed over the network to each client every time they are modified. This access may be synchronous, meaning that a table update provides not only the new contents of the table, but also the event which signals the time it happened. In the PS timing system, such events can be used to transmit accelerator timings to application programs, so events are often more important than the new table contents.

[1] On leave from IHEP Protvino, Russia

Figure-01 Network layout

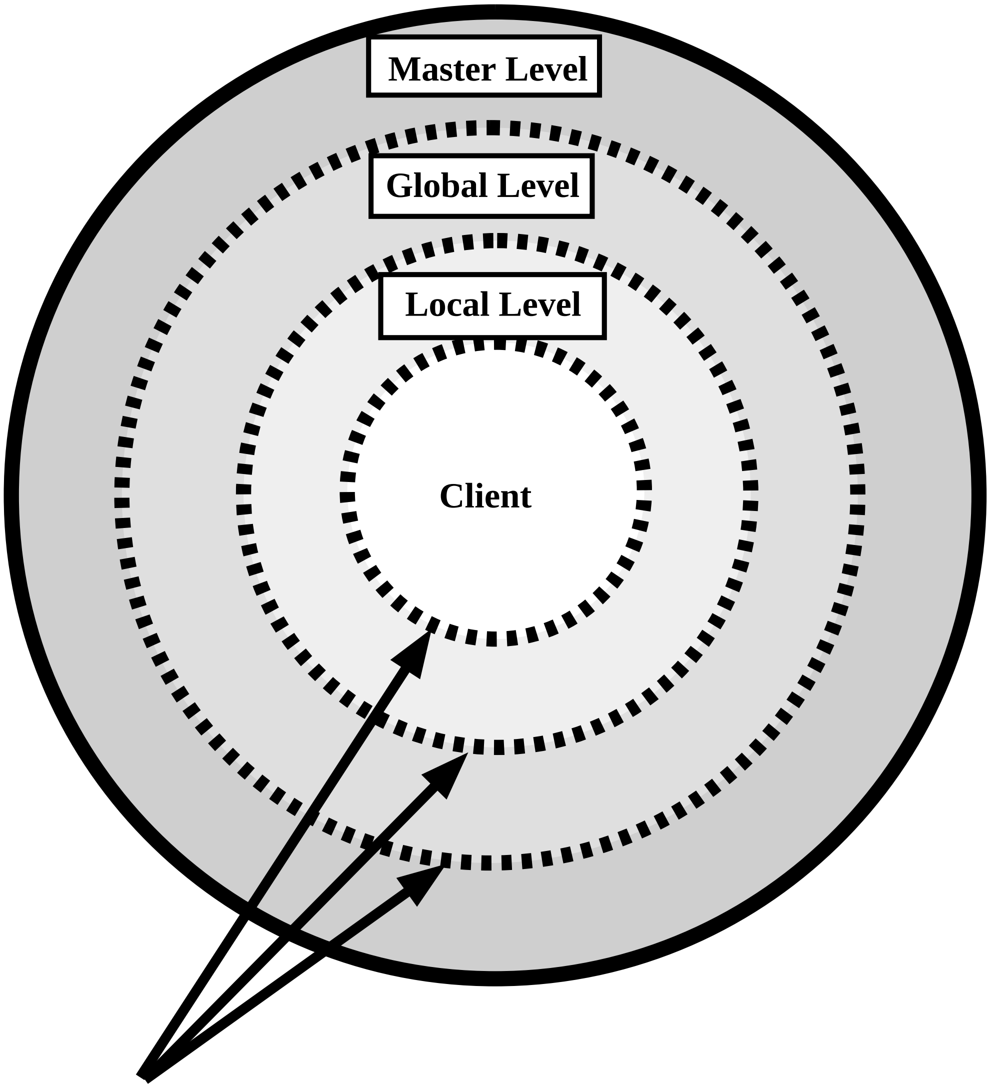


The concept of distributed memory satisfies these requirements, in which all copies of a memory segment residing on a set of *registered* hosts contain the same data and hence all writes into this memory space must be distributed across all copies. In DTM, this is a *One-To-Many* data flow, where a single client non-explicitly provokes a table change on all other registered clients in a heterogeneous network. This is not the only data flow; in *Many-To-Many* the consuming clients sequentially processes a queue of incoming new table instances. In a third data flow, one client explicitly reads another client's copy of a non-distributed table, this *One-To-One* flow has some similarities with a classic remote-procedure-call mechanism. All three of these mechanisms are used by the PS timing system and DTM has greatly simplified the task of implementing them, by providing applications with a fast platform-independent method for obtaining the data they need. Application programs are relieved of the burden of initializing and maintaining up-to-date copies of volatile data and of knowing who else is using it. No explicit inter-client actions are required, as all data is distributed between them by the DTM system. System maintenance is easier because of the ease with which data-driven programs can be built and the ease with which they can be distributed over a heterogeneous network. Lastly, a faster and more efficient use of limited network resources can be achieved by avoiding unnecessary transactions such as polling.

2. THE IMPLEMENTATION

The DTM system is built on a hierarchy of three classes of server daemon [Fig. 2]; a unique master server runs on the backbone subnet, one global server runs on each control subnet and one local server runs on all other hosts. Except when a network broadcast is used, all DTM data tables flowing between clients pass via the master daemon along the path shown in Fig. 3.

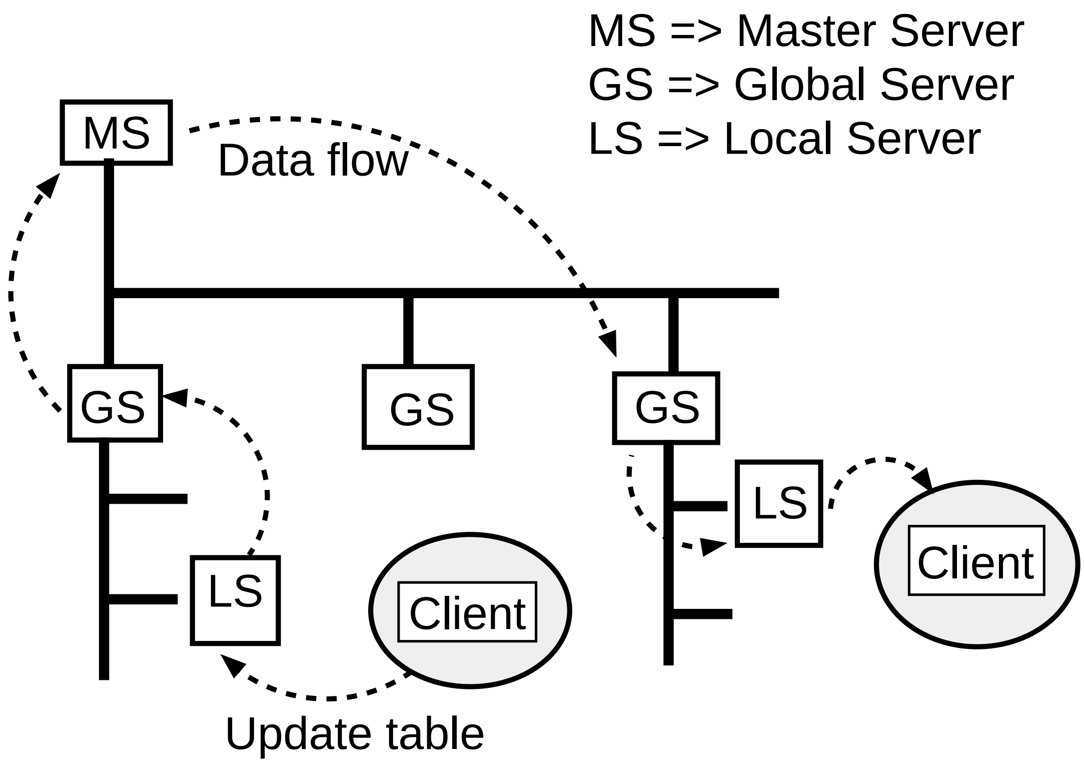
Figure-02 Daemon and client communications hierarchy



These communication layers export the services provided by the next level up to the network, making all internals available for diagnostic tools. Local function calls are used between the levels in a single server daemon.

The class hierarchy starts with the local server, which provides the applications program interface, and whose functions are available on all hosts. Next is the global server which inherits the local server functions and provides specific global functions. The unique master server inherits all local and global functions and provides specific master functions. Thus only one server runs on any given host. All communications between class levels and between the client and server are available across a communications layer which renders all such transactions independent of their location on the network. This approach has greatly eased the task of building diagnostic tools, which are able to communicate with any level of any daemon anywhere in the network. The current implementation makes use of UDP and UNIX domain sockets for inter host and inter task transactions. All daemon launches are triggered automatically in the standard way by *inetd*. Packet splitting, sequencing and error correction with retry are handled and connections are dropped and re-established between daemons using time-outs and an*I-am-alive* protocol. When a control subnet is isolated from the backbone, all DTM table updates to or from that subnet will be completed later automatically, when the inter server communications are re-established via the*I-am-alive* protocol..

Figure-03 DTM server daemon data flow



For ease of maintenance, the only system knowledge given to a server on start up is the directory path name: */usr/dtm*, where it finds all the information it needs in two configuration files. This directory is usually an NFS mount point for local server hosts on their global server and, besides the configuration files, it also contains disk copies of tables in network format and client registration tables. The two ASCII configuration files describe the attributes of each table and the host names of the subnet global server and master server respectively. The attributes of a table are as follows:·

• The tables name:	E.g.	CPS_LINE_NAMES·
• The tables format specification:	E.g.	1L32{1L24{9C51C}}·
• The table producer host list:	E.g.	PsStation1, PsServer19·
• The table properties:	E.g.	PROD+NONC

The table name is any string by which the table is identified, and provides the binding between DTM servers for its transfer in the network. The table format string describes the table structure. The above example means: "*1 Long, 32 Structures of { 1 Long 24 Structures of { 9 Characters, 51 Characters } }*". The producer list contains the list of host names permitted to update the table. The table properties are any coherent subset of the following:·

- NONC Not kept on disk, so the table exists only in memory. This property concerns tables which are updated frequently. Disck-kept tables are only useful when recovering from a general power failure, or when bringing up a subnet in isolation, as in all other cases the latest table contents are transferred between servers when communications are re-established via the *I-am-alive* protocol..·

- LOCL The table exists in local heap memory and not in shared memory. In this case the table memory is allocated to the client via the application program interface library.

- PROD The table may only be updated by a host in the producer list. Any write attempts by clients not on the producer list result in an error return from the library.·

- MCOP The table has multiple different values. Used in *Many-To-Many* and *One-To-One.*·

- FIFO, FILO, LIFO ... Various table queue disciplines for *Many-To-Many.*·
- BRDC The new table contents is broadcast, rather than using inter-server point-to-point transfers. A UDP type-C subnet address is currently used, but for hosts outside the type-C subnet, the table is transferred in the usual way.

3. THE APPLICATION PROGRAM INTERFACE

The entire DTM application program interface is described by 10 simple function calls. All other daemon inter-level function calls are also available, but they are used exclusively for system internals and specialized diagnostic programs and are not listed here.

3.1 Basic One-To-Many non-synchronous access

For the more common *One-To-Many* data flow the following five very simple functions provide an elegant way for an application to use non synchronized access to distributed shared memory.·

- int **DtmrtRegisterTable**(table_name)

  A client calls this function when it wants to register its interest in a table. Two registrations are actually made by the system, a global registration saying that this host is using the specified table and a local registration saying that this task PID uses the table. Hence for any given table, there is one global registration per host using it and one local registration for each PID on that host using it. When the last local registration is removed, the global registration is also removed and all system resources are de-allocated.·

- int **DtmrtUnregisterTable**(table_name)

  A client may call this function if it is no longer interested in a table. The local DTM server in any case checks that the PID exists each second and will clean up its registrations within this time if the task stops running for any reason. The only use of this function is thus in limiting the total resources consumed by a client at any one time, in particular the shared memory attach count.·

- table * **DtmrtShareTable**(table_name)

  This function returns a pointer to the registered table either located in shared memory, or on the local heap allocated within the function. Typically a client will cast the returned pointer into a defined data structure; for reading it can be directly accessed and for writing, if it is located in shared memory, a copy should be made because DTM can update it at any time.·

- int **DtmrtUpdateTable**(table_name, new_table)

  If this host has the correct write permission, calling this function pushes the new table contents out to all registered clients on the network and overwrites the local copy.·

- int **DtmrtCheckTable**(table_name, callback)

  This routine can be called periodically from an application program main loop. If the table has been updated, the callback is invoked with the name of the table as its parameter. In this way an application is able to take specific action each time the table is changed; this is the non-synchronized access method. If the table name is NULL the callback is called once for each updated table.

3.2 Other data flows Many-To-Many and One-To-One

The next two function calls provide support for the *Many-To-One* and *One-To-One* types of data flows:, ·

- int **DtmrtRegisterTableCopies**(table_name, queue_size)

  This function is similar to the normal registration, but in addition, the DTM server allocates enough memory to contain *queue_size* copies of the table, which it will pass back one at a time to the client in the event of multiple updates. This type of registration thus provides the support required for *Many-To-One* data flows.·

- int **DtmrtReadRemoteTable**(remote_host_name, table_name, table_buffer)

This function explicitly reads the specified table from the remote host into a supplied table buffer. No host, in this case, would call DtmrtUpdateTable, but instead clients write directly into the table in shared memory. When many remote hosts do this, then the table instances all contain different data which can be read via this function. This function thus provides the support required for *One-To-One* data flows

3.3 Synchronized access and event handling

Synchronized table access is based on subscribing to table update events, which are interesting in themselves as a means of sending real timing events over the network, in this case, the time of arrival of an update event is often more important than the table contents.·

- file_descriptor **DtmrtSubscribeTable**(table_name)

Subscribe to real time table update events, the function returns an integer file handle suitable for use within a select system call. The table must already be registered.·

- int **DtmrtUnsubscribeTable**(table_name)

Un-subscribe to the table update events, this will happen anyway if the subscribed task stops running for any reason.·

- int **DtmrtSelect**(select_mask, event_handler, time_out)

In addition to calling the standard UNIX file select system call, the table data is transferred between the server and the client's local memory. A subsequent call to Dtmrt-CheckTable will provide the table name.

4. CONCLUSION

The latest DTM package has been operational in the PS control system since March 1995, and is dealing with 20 different tables distributed over 6 control subnets, and about 1000 global registrations is a typical figure. It is currently running on:·

• DEC MIPS	***Ultrix 4.4·***
• DEC ALPHA	***OSF/1 2.1·***
• IBM RS/6000	***Aix 4.1·***
• MOTOROLA MVME147/167	***LynxOS 2.2·***
• CETIA PowerPC 601	***LynxOS 2.2·***
• PC486	***LynxOS 2.2·***
• SUN Solaris 2.3	***SunOS 4.1.3·***
• HP	***HP/Ux 9.1***

During this time it has recovered from network failures, CPU crashes, general sector power-failures, task crashes and new users developing application programs. The simple and elegant application program interface has permitted the *port* of complex timing-system libraries and the applications using them, to all of the above platforms with minimal effort. DTM also allows us to modify control system tables on all platforms while applications are still running and avoids the need to build global data into these programs, with the maintenance problems and inflexibility this causes.

REFERENCES

[1] J. Lewis, V. Sikolenko: "The new CERN PS timing system", ICALEPCS, Berlin, Germany Oct 18-23, 1993, Nucl. Instr. And Meth. A352 (1994), 91.

[2] F. Perriollat, C. Serre: "The new CERN PS control system overview and status", ICALEPCS, Berlin, Germany Oct 18-23, 1993, Nucl. Instr. And Meth. A352 (1994), 86.