Data Migration Manager (DMM) Functional Specification Document

Open File and Storage Technologies (OFAST)

Rev	Author	Date	Description of Changes or Status
1	Mark Cruciani	01/07/08	Initial draft
2	Steve Larson, David Villineau, Erick Liska, Roy Lee	02/20/08	First cut at architecture descriptions for Section 4.
3	Mark Cruciani	04/21/08	Collapsed 3 sections into a single user-space section. Incorporated many 20 question anwers into the appropriate places.

Table of Contents

1 Product Description

The DMM project is developing an open-source tool, DMM, designed to facilitate the movement of file systems from one platform to another over the NFS and/or CIFS protocol. A key feature of DMM is that applications can continue to read and write to file system content while the migration is running.

One or more file systems can be migrated from their read-only sources, to a target file system resident on the platform where DMM is hosted. The user is given full control of all aspects of migration (resources consumed, order of migration, etc) via a GUI.

Progress of the DMM project is being tracked via the PSARC/2007/692 case.

1.1 Definition

The DMM data migrator is intended to be a tool used by system administrators or knowledgeable end-users to move one or more file systems from a read-only source platform to the target platform on which DMM resides. File meta-data and data is moved over both the NFS (v3/v4) protocol and the CIFS protocol as defined by the Solaris CIFS client (PSARC/2005/695) DMM will allow specification of a single protocol for migration in cases where files are not associated/accessed by both.

Files are migrated both on-demand as a result of a processes access or asynchronously under the control of an migration schedule.

DMM will be controlled by either a GUI or a CLI where the CLI will be a proper subset of the controls permitted by the GUI.

DMM is structured as two main components, user-space and kernel:

The user-space component contains the majority of the product and is responsible for the effort required to migrate a file system. It is designed to be scalable in controlling many simultaneous file system migrations. It's communication to the kernel portion is via an ioctl as well as through the standard POSIX I/O interface.

The DMM kernel component is lightweight and intended to be a means of controlling file system activity through the use of Solaris FEM monitors attached to vnodes in the target file system. Files (vnodes) in the target file system are monitored via this process from their namespace inception through the time required to fully migrate all data and meta-data. Activity on a monitored file results in an event being propagated to the user-space component for action via one of several Solaris door calls per file system being migrated.

All state information associated with a migration is tracked in a database and DMM is able to fully recover an active migration in the event of any number of failures or crashes.

[MARK] SHOULD HAVE A DIAGRAHM HERE

1.2 Motivation, Goals and Requirements

Today, customers choosing to adopt next-generation storage solutions have no easy method of transferring data onto these new systems while retaining access to these data by applications.

A dependable, easy to use migration solution which transmits all aspects of user data greatly simplifies this task and reduces the costs associated with transition.

1.3 Changes From the Previous Release

N/A

1.4 Program Plan Overview

1.4.1 Development

Please see the DMM Development Plan which will be available on the DMM OpenSolaris project page at http://opensolaris.org/os/project/dmm/

1.4.2 Quality Assurance/Testing

Please see the DMM Test Plan which will be available on the DMM OpenSolaris project page at http://opensolaris.org/os/project/dmm/

1.4.3 Documentation

Please see the DMM Documentation Plan which will be available on the DMM OpenSolaris project page at http://opensolaris.org/os/project/dmm/

1.4.4 Release Cycle

1.4.5 Technical Support

1.4.6 Training

1.5 Related Projects

1.5.1 Dependencies on Other Sun Projects

Solaris CIFS client (PSARC/2005/695), mySQL 5.1

1.5.2 Dependencies on Non-Sun Projects

N/A

1.5.3 Sun Projects Depending on this Project

Appliance developer kit, future Sun NAS appliances.

1.5.4 Projects Rendered Obsolete by this Project

Sun Storagetek Client Services NT and NFS Migrators

1.5.5 Related Active Projects

See above.

1.5.6 Suggested Projects to Enhance this Program

1.6 Competitive Analysis

Sun today has two individual migrators, each responsible for a single protocol. These are the NT and NFS migrators and their use is coordinated via a client solutions engagement. Some of the limitations of these are that they are obviously two separate tools that must be used in an either/or fashion as well as the fact that they do not have an in-kernel component which requires them to use a polling mechanism from user-space (often from another intermediary host) to determine when migration is complete.

2 Technical Description

2.1 DMM Component Specification

The Data Migration Manager consists of several components, including a kernel-mode pseudo driver, multiple user-mode programs and libraries, as well as a GUI management application and a CLI management command.

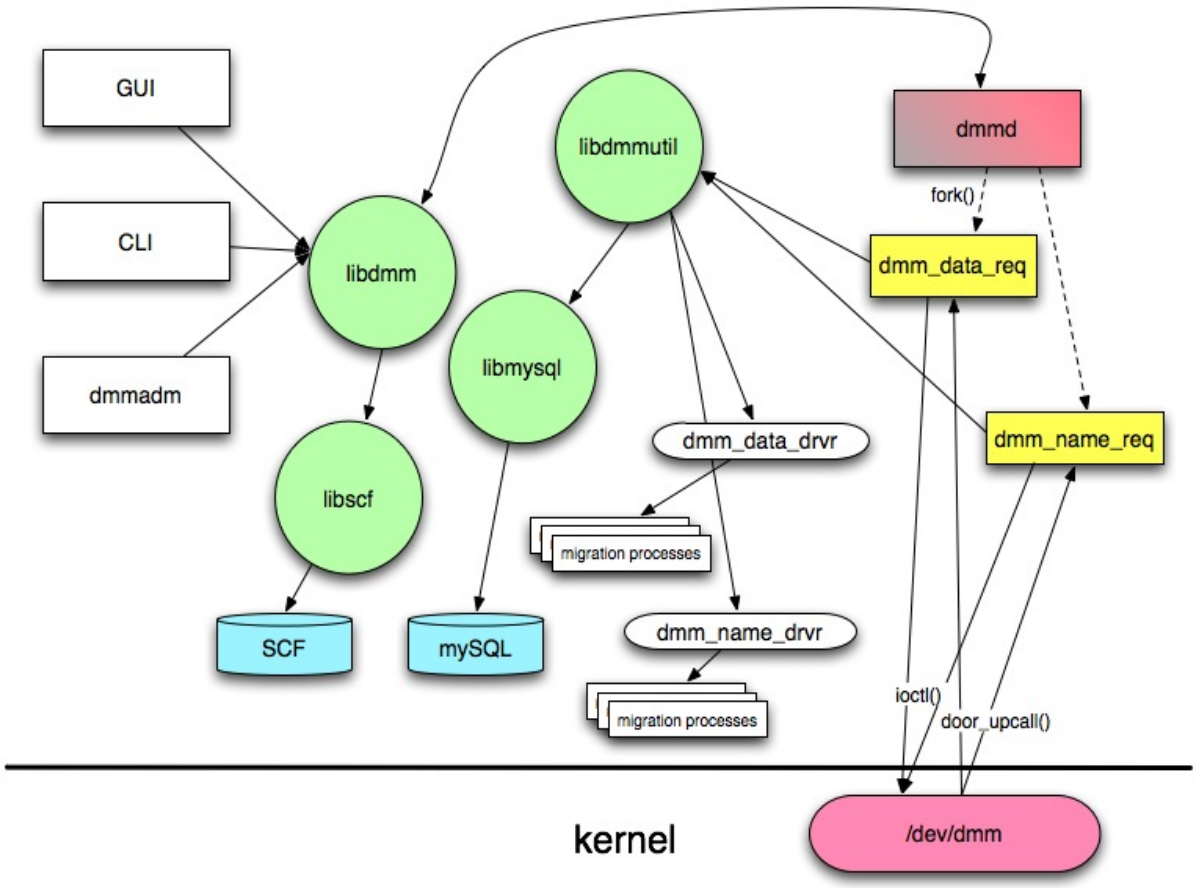


Illustration 1: DMM Module Interfaces

2.2 DMM API (Management interface)

The DMM management API is implemented in *libdmm*. DMM maintains some basic information, such as the DMM database path, in SCF. All information pertinent to actual file system data migration is maintained in the DMM database, the API for which is implemented in *libdmmutil*.

Data migration management is centered on individual migrations, where a migration is defined as the transfer of a complete namespace within one file system to another file system. The DMM management API is based on this migration definition; migrations can be created, destroyed, started, stopped, and modified. All requests to create or change the state of a migration are communicated to the DMM service daemon through door calls. All requests for migration state are obtained through calls to the DMM database API in *libdmmutil*.

At the root of the migration functionality is the DMM service daemon, *dmmd*. The service daemon is implemented as an SMF service. By default it is disabled, but can be enabled using *svcadm* (1M). The daemon is responsible for receiving and dispatching migration requests. Requests are received via the API door server, and dispatched to migration child processes as program arguments.

The actual migration is handled by processes spawned by the daemon; *dmmnamed* for namespace migration, and *dmmfiled* for file migration. These processes in turn delegate the bulk of the work of copying file system objects to another process, *dmmworkd*. All four programs use some common functionality, including the DMM database interface, implemented in *libdmmutil*.

2.3 Interfaces

2.3.1 Exported Interfaces

None.

2.3.2 Imported Interfaces

FEM, Solaris doors, VFS/vnode, mySQL

A complete list to be provided here.

2.4 User Interface.

2.5 Compatibility and Interoperabi lity

2.5.1 Standards and Conformance

2.5.2 Operating System and Platform Compatibility

2.5.3 Interoperability with Sun Projects/Products

2.5.4 Interoperability with External Products

2.5.5 Coexistence with Similar Functionality

2.5.6 Support for Mul tiple Concurrent Instances

DMM is installed via the packaging facility. It consists of two packages:

SUNWdmmr DMM Service Root package: Components installed under "/" directory
SUNWdmmu DMM Service User package: Components installed under "/usr" directory

For each package, there are 3 sections identified:

- Components installed common to all platforms
- Components installed specific to x86 platforms
- Components installed specific to SPARC platforms

3.1.1.1 SUNWdmmu

Components common to all platforms:

(These path names are not final)

/usr/dmm/ bin/dmmd
Userland DMM service daemon

/usr/dmm/bi n/dmmfi led
/usr/dmm/ bin/dmmnamed
/usr/dmm/ bin/dmmwork d
Userland migration processes private to the service daemon

/usr/dmm/ bin/dmmadm
Userland DMM service configuration CLI

/usr/dmm/ lib/l ibdmmut il.so
Userland DMM service common functionality, database interface

/usr/dmm/ lib/l ibdmm.so
Userland DMM service API

Components specific to x86 platforms: None
Components specific to SPARC platforms: None

3.1.1.2 SUNWdmmr

Components common to all platforms:

/var/svc/manifest/system/dmm.xml
DMM service manifest file to start dmmd as an SMF service

Components specific to x86 platforms:

/kernel/drv/ dmm
DMM kernel pseudo driver for the 32-bit x86 platform

/kernel/drv/amd64/ dmm
DMM kernel pseudo driver for the 64-bit x86 platform

Components specific to SPARC platforms:

/kernel/drv/sparcv9/ dmm
Kernel DMM pseudo driver for the 64-bit SPARC platform

3.1.1.3 Additi onal Components

Manpages: dmmd (1M), dmmadm (1M), additional manpages TBD.

3.1.2 Effect on External Environment

3.2 Install ation

3.2.1 Installation procedure

3.2.2 Effects on System Files

3.2.3 Boot- Time Requirements

3.2.4 Licensing

3.2.5 Upgrade

3.2.6 Software Removal

3.3 System Administrati on

4 GUI Architecture

This section gives a detailed architectural overview of the DMM GUI.

4.1 Descri ption

The DMM GUI is web-based and runs inside web browsers. User can access the GUI through a network-enabled client and connect to the DMM host server through a set of supported web browsers. No installation is necessary on the client.

Through the GUI, user can initialize, control and monitor the data migration process.

4.1.1 User Audience

The anticipated users are system administrators who needs a file system independent mechanism to migrate data from existing hardware to new hardware with minimal to no downtime.

A typical usage pattern: System administrator configures DMM to migrate data from the source to the destination. Start the migration process and leave it. Administrator comes back from time to time to monitor the progress. Depending on system performance and load, he/she may choose to increase or decrease system resources allocated to this task. When finished, system administrator performs the post-migration clean-ups.

4.1.2 User Model

4.1.2.1 Data Source

Data Source is the source file system where data will be migrated from. It can be

mounted or unmounted and shared using NFS or CIFS.

4.1.2.2 Data Target

Data Target is the target file system where data will be migrated to. It can be mounted or unmounted and shared using NFS or CIFS.

4.1.2.3 Data Access Point

Data Access Point is the temporary file system where data could be access during data migration. If specified, this file system can be mounted and accessed through NFS or CIFS.

4.1.2.4 Data Migrator

Data Migrator is the actual running process that handles the data migration task. The migrator has the following stats:

Stop – The migrator is suspended, taken out of memory and stored on disk.

Halt/Remove – The migrator is terminated and the data on the data target is removed.

Halt/Keep – The migrator is terminated and the data on the data target is preserved.

Pause – The migrator is suspended in memory.

4.1.2.5 Data Migration Manager

Data Migration Manager provides the managing and monitoring interfaces for data source, data target, data access point and data migrator. Its functionality includes:

Initial ization – Specifying data source and data target. Mounting data source as read-only and create data access point.

Management – Stop/Pause/Halt individual migrators. Modifying migrator load factor schedule, change migration order and final post migration clean up.

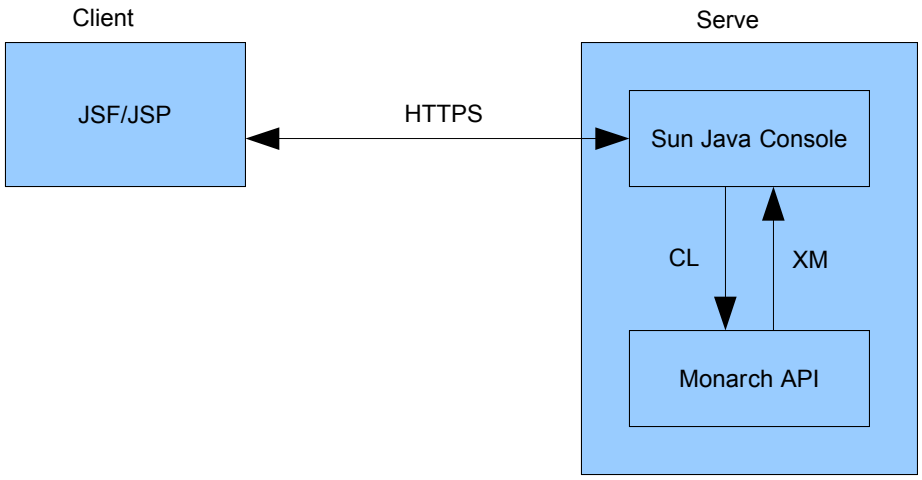
Monit oring – Provide migrator related information such as migration speed(Bytes/Sec) and stat. Display error notifications from the migrator and keep a log of running and finished data migrations.

Error Recovering – Allow accessing to the underlying database for certain error recovering.

4.1.3 UI Toolkit/Framework

There will be a web-based DMM GUI implementation for OpenSolaris which will be based on Woodstock.

The Woodstock UI Components are based on Java Server Faces. The protocol used between the client and the server would be HTTPS through an application server. The functions would be accessed through the CLI while the results would be returned in XML.



4.1.4 Localization & Internationalization

Specific localization requirements are undetermined; however, internationalization support will be implemented. The internationalization will focus on the text only. Layout, icons and colors will be implemented as regional neutral as possible.

4.1.5 Operating Environment

Any operating environment that is capable of running any of the specified web browsers is supported.

4.1.6 Security, Encryption and Authentication

User authentication will be provided by Sun Java Web Console. The default encryption is SSL.

4.2 Operati on

This section describes DMM GUI's operating behaviors and requirements.

4.2.1 Start/Stop

The DMM GUI is started when user establishes a session with the application server through a web browser. The session/DMM GUI can be stopped by either logging out or closing the web browser.

4.2.2 Operating Requirements

The Http server and application server must be running on the server side to take client requests. Clients must use compatible web browsers for access. Network connection is required if accessing the server from a remote client.

4.2.3 Device Interaction

The DMM GUI interacts with other DMM modules through the underlying DMM CLI which is defined in detail in the CLI section.

5 DMM Command Line I nterface (CLI)

This section provides details of the Command Line Interface (CLI) for the migrator service.

This document defines the migrator service command line interface. The migrator service CLI comprises the following commands:

- dmmadm (1M) Data Migration Manager service configuration utility
- dmmd (1M) Migrator daemon

5.1 dmmadm

NAME

dmmadm - data migration manager service configuration utility.

SYNOPSIS

```
dmmadm create [-h] [-p property=value]
dmmadm delete [-f] migid
dmmadm start migid
dmmadm halt migid
dmmadm pause migid
dmmadm set -p property=value [-p property=value]... migid
dmmadm get [-p property=value]... [migid]
dmmadm list
dmmadm show
dmmadm stat migid
```

DESCRIPTION

The dmmadm command is used to manage data migrations. A migration is defined as the movement of a complete file system to a new namespace. Migrations are identified by a unique identifier, the migid.

The dmmadm command provides the ability to create, delete, start, and halt migrations, as well as the ability to get and set migration

properties.

Some properties have default values; others must be specified at the time the migration is created. For example, the migration source file system and target file system values must be defined with the create subcommand.

Subcommands

The dmmadm command supports the following subcommands:

create [-h] [-p property=value]
: Create a new migration. If the -h option is specified, the migration is created in a halted state; a subsequent start command is needed to begin the migration.

delete [-f] migid
: Delete an existing migration.
The -f option forces a running migration to be deleted.

start migid
: Start or resume a halted migration

halt migid
: Halt a migration

pause migid
: Pause a migration

set -p property=value [-p property=value]... migid
: Set the values of migration properties

get [-p property=value]... [migid]
: Get the values of migration properties

list
: List all existing migrations

show
: Display detailed information for all existing migrations

stat migid
: Display statistics for a migration

MIGRATOR PROPERTIES

segsize — Migration segment size.
The value of segsize is a string
with a numeric component and an optional letter component
to specify a unit size, in the format "N[.N][KMGTP][B]".

Following the numeric component, the optional unit can be
specified as either one or two characters; e.g., either 'K'
or 'KB' to specify kilobytes. Unit specifiers are not
case-sensitive, and must follow the numeric value
immediately, with no whitespace between the two. With
either no unit specifier, or a unit specifier of only 'B',
the numeric value is assumed to be in bytes.

source — Source file system.

target — Target file system.

protocol — Client protocol. Valid values are "nfs", "cifs", and "posix". The default value is "posix".

collision — Name collision policy. Valid values are "overwrite", "rename", and "error". The default value is "error".

logfile — Full path to migration log file.

maxnsproc — Maximum number of namespace (directory tree) migration processes that can be executing at one time.

resnsproc — Number of reserve namespace (directory tree) migration processes.

maxdfproc — Maximum number of data (file) migration processes.

resdfproc — Number of reserve data (file) migration processes.

EXAMPLES

```
# dmmadm create -p source=/mnt/oldhostA/home -p target=/home

# dmmadm delete 1

# dmmadm set -p segsize=256m 1

# dmmadm start 1

# dmmadm create -p source=/mnt/oldhostB/work -p target=/work
```

-p segsize=2G

dmmadm halt 1

dmmadm set segsize=512m logfile=/var/dmm/oldhostA 2

EXIT STATUS

0	Command completed successfully.
non-zero	Command failed with error.

ATTRIBUTES

See the attributes(5) man page for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWdmmu
Utility name and options	Evolving
Utility output format	Not-An-Interface

SEE ALSO

dmmd(1M)

5.2 dmmd

NAME

dmmd - data migration manager service daemon

SYNOPSIS

/usr/dmm/bin/dmmd

DESCRIPTION

dmmd is the daemon that handles data migration. A migration is defined as the transfer of an entire namespace within a single file system, referred to as the source, to another file system, referred to as the target.

The dmmd service is disabled by default, and can be enabled using svcadm (1M).

OPTIONS

None.

OPERANDS

None.

EXIT STATUS

0	Daemon started successfully.
non-zero	Daemon failed to start.

FILES

/usr/dmm/bin/dmmd	data migration manager daemon binary

ATTRIBUTES

See attributes(5) for descriptions of the following attributes.

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Utility name and options	SUNWdmmr
Interface stability	Evolving
Utility output format	Not-An-Interface

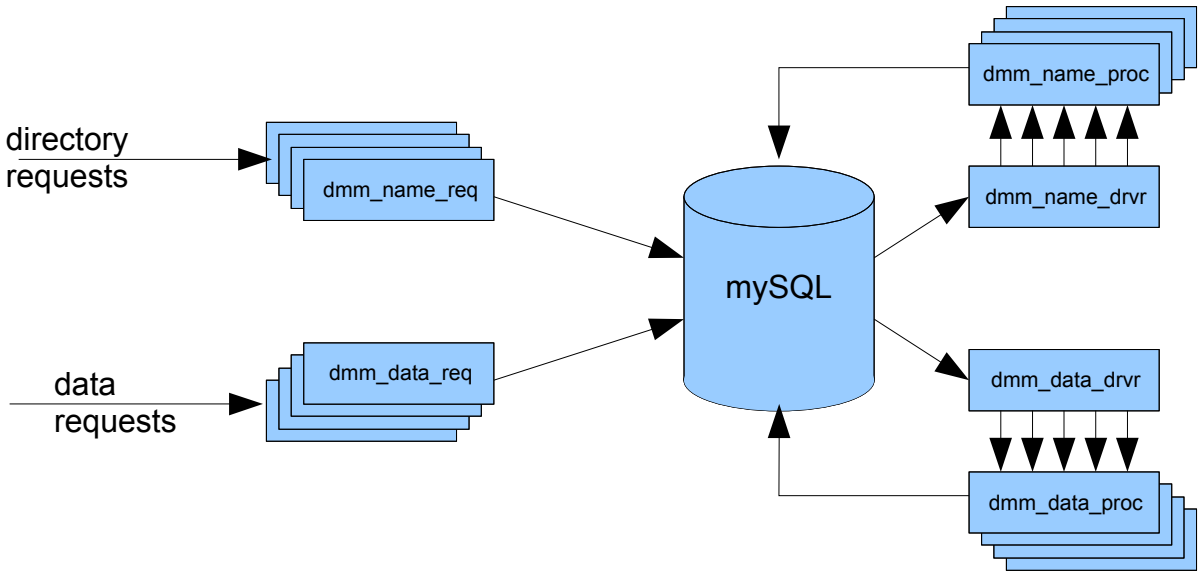
SEE ALSO

svcs(1), svcadm(1M), dmmadm(1M)

6 User-space Mi grator Component

The user-space portion of the migrator performs all of the data movement, controls all client access and makes all policy decisions. It relies heavily on the kernel component to intercept client access requests and pass these up into user-space via Solaris doors. Upon notification of these client requests the migrator will attempt to immediately migrate the file or directory to satisfy the request. The migrator is interested in three basic types of access requests: directory requests, file requests and requests which modify only the name-space (rmdir, rm, etc)

Scalability is one of the primary goals of DMM's design. Each migration task has its own database and a set of dedicated processes. The following diagram illustrates a single migration task:



Access requests are intercepted via FEM monitors which result in door up-calls to the request processes (name_req and data_req) which make updates to the database. The driver processes (name_drvr and data_drvr) retrieve records from the database that need to be processed immediately and dispatch the request to the worker processes (name_proc and data_proc).

Each of these requests perform appropriate actions on the central database and may suspend until a migration completes or return immediately if the migration has already completed. Note that monitors are removed as soon as possible to eliminate any unnecessary overhead in the client access path.

6.1 Directory requests

Directory requests (name lookups) are intercepted by a FEM monitor and notification is sent up to user-space (via a door up-call) so that the migrator can add an entry to the database indicating that a directory requires migration. Since the thread doing the lookup and all others are blocked while the directory is being migrated, the door-up call assigns a high priority to the database entry. A portion of the migrator, name_drvr, extracts database entries in priority order and for each entry, migrates/populates all objects within that directory.

Population of each directory object entails the creation on the target file system of an empty object (and vnode with attached FEM monitor) of the same type as the source file system. The meta-data associated with the original object on the source is either added to the object or locally attached in the monitor space (size for example) for use in satisfying references over the duration of the migration. After all objects within the directory has been populated, the FEM monitor attached to the vnode of the directory itself is demoted to a lesser role of tracking namespace changes. This persists until all objects below that point in the directory structure have been migrated whereupon the directory FEM monitor is removed.

During the population process, objects which are directories are recorded in the database for future directory migration. Those which are regular files are recorded in the database for future data migration. Most special files are created but require no further processing. Symbolic links for example that are relative to the source target could be changed to be relative to the target with interaction from the user. DMM will evaluate these possibilities over the course of development.

6.2 File request s (data migration)

In a process similar to how directory requests are handled, file requests are intercepted by a FEM monitor and sent up to user-space via a Solaris door-up call (separate door to a separate process) Recall that each file placeholder was created at the time its parent directory was populated.

The data_req process at the top of the door, adds an entry for each file to the database for an eventual removal by a data_drvr process. Upon database removal, a file is copied by a data_proc process from the source over one or more protocols, and written to the target file system. Upon completion, the data_proc process removes the FEM monitor by issuing a call though the DMM kernel module's ioctl.

There will be several places where optimizations will be used. A file's access pattern may increase the priority associated with that file's migration in the database (ie, 4 readers blocked will result in that file being migrated sooner)

Another optimization centers around minimizing latency to waiting reader/writer threads by fetching I/O in the smallest amount required to satisy the request.

The data_proc processes which do the actual copying for the migration, do I/O in *segment* sized units. The segment size is tuneable but may also change dynamically based upon the file size.

Once the segments that span a VOP_READ request for example, have been obtained (in fact, multiple segments of the same file may be migrated in parallel if multiple requests occur on the same file) a thread can be released.

6.3 Name-space modifi cation requests

The last type of request is the name-space request (e.g. rename, remove, rmdir) which causes a change in the name-space that the migrator may have to resolve. For example, if a file is renamed (perhaps even relocated to another directory) prior to it being migrated then the migrator needs to know the new location so the data migration process can write the data blocks to the correct file.

Once a directory has been initially populated, the FEM monitor on that directory is demoted to what's terms a *lightweight monitor*. This monitor remains active on the directory until all child files and directories below have been migrated.

Lightweight monitors simply allow namespace transformations to be tracked in the database.

6.3.1 Phased commits

Certain operations like namespace modifications will require DMM to use a phased commit approach when tracking these changes in its database.

This protects the integrity of the target file system and guarantee the restartability of the migration process. We will rely on the database to determine how to restart the migration should an interruption occur.

The first phase (intent) occurs prior to the creation of an object in the target file system. An entry for the object is added to the database to record the intent to modify the target file system. Any time after this point the database will accurately track the exact state of the object.

The second phase is where the object is actually created in the target file system. In the event of a catastrophe this operation can be restarted in the event it was not recorded properly in the file system.

The final phase updates the entry in the database to properly reflect the state of the creation. After this point the object creation is complete and will never need to be restarted.

6.4 DMM Source file system access layer (I/O layer)

DMM's user-space portion will utilize a discrete component to access a source file system for migration purposes. This component will be configurable for each source file system. Multiple "plugins", distinguished between protocols, protocol versions or dialects, or even in-kernel or user-space protocol implementations, will be available.

6.4.1 Description

This component isolates protocol-specific concerns from DMM. It also provides the ability to configure specific protocol access at runtime as well as a framework whereby additional plugins can be developed.

Each protocol is assigned a plugin for each migration, but multiple simultaneous file system migrations can each use a different plugin for a given protocol.

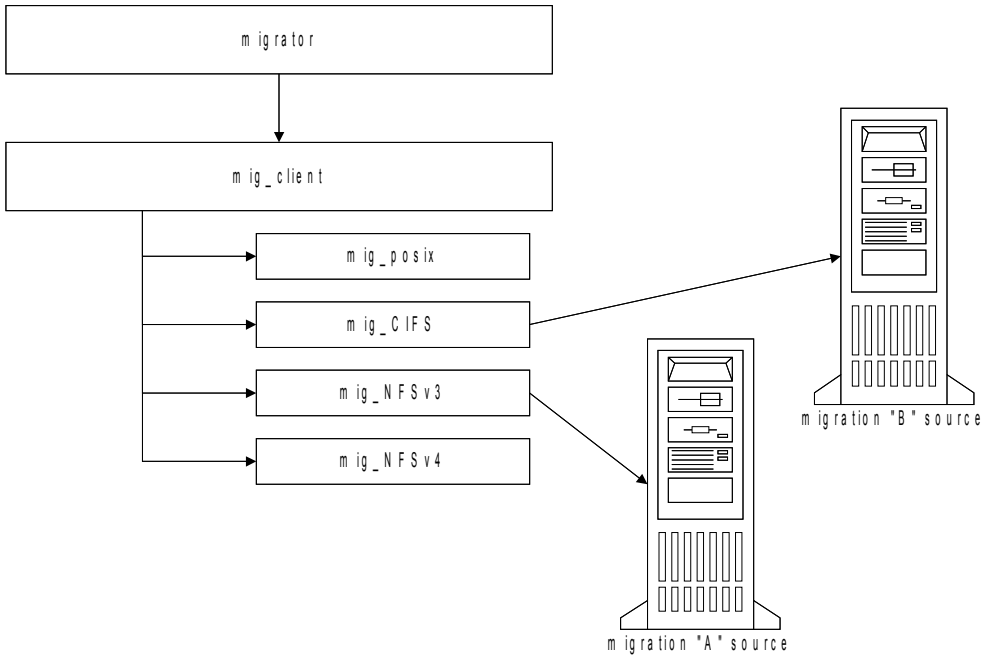


Illustration 2: Source File System I/O Component and Plugin Overview

The purpose behind this component is twofold: where applicable, to increase performance by avoiding overhead and using protocol-specific functionality directly, and to enable protocol-specific data to be migrated. In the absence of plugin support for a specific source file system, a generic POSIX-based I/O plugin will be used.

6.4.2 Data Structure

```
typedef enum
{
      MIGCL_POSIX,
      MIGCL_NFSV3,
      MIGCL_NFSV4,
      MIGCL_CIFS
}
migclient_t;
```

6.4.3 Interfaces

6.4.3.1 External

```
int client_set(int migid, migclient_t client);
```

Defines the client type for the specified migration and loads the corresponding client plugin.

Parameters:
- migid: Migration id for which the client will be defined.
- client: Client type definition.

Return Values:
0 on success, errno values on error.

```
int client_get(int migid, migclient_t *client);
```

Retrieves the client type for the specified migration.

Parameters:
- migid: Migration id.
- client: Pointer to client type. The client type for the specified migration is returned here.

Return Values:
TBD

```
int cl_open();
```

TBD

Parameters:
TBD

Return Values:
TBD

```
int cl_close();
```

TBD

Parameters:
TBD

Return Values:
TBD

```
int cl_read();
```

TBD

Parameters:
TBD

Return Values:
TBD

```
int cl_getacl();
```

TBD

Parameters:
TBD

Return Values:
TBD

```
int cl_getxattr();
```

TBD

Parameters:
TBD

Return Values:
TBD

```
int cl_opendir();
```

TBD

Parameters:
TBD

Return Values:
TBD

```
int cl_readdir();
```

TBD

Parameters:
TBD

Return Values:
TBD

```
int cl_closedir();
```

TBD

Parameters:
TBD

Return Values:
TBD

6.4.3.2 Internal

TBD

6.4.4 Operation

This component is structured as a library. Each protocol plugin is also structured as a library. While this component recognizes available plugins at runtime, if no compatible ones are detected it defaults to a POSIX-based I/O plugin.

7 Kernel Migrator Component

This section describes the overall architecture of the kernel component of DMM that is responsible for client process blocking and communication with the user-space components.

7.1 Descri ption

The file system Monitor (“monitor”) is responsible for intercepting and temporarily blocking file system operations made by client applications. Once intercepted the monitor will notify the migrator that an operation is pending for a directory or file, and therefore an immediate migration should be scheduled.

File system monitoring will be accomplished using the FEM (PSARC/2003/172) API to install, manage and remove monitors. Three different types of monitors will be used:

7.1.1 Directory monitor

Used to monitor directory-level operations. Once triggered the monitor will inform the migrator that a namespace migration is required. Operations of interest include: VOP_LOOKUP, VOP_OPEN, VOP_CREATE, VOP_GETATTR.

7.1.2 File/entity monitor

Used to monitor file system operations to a non-directory file system object. Once triggered the monitor will inform the migrator that a data migration is required. Operations of interest include: VOP_GETATTR, VOP_OPEN, VOP_READ, and VOP_WRITE.

7.1.3 Namespace monitor

Used to monitor all file system operations that impact an already-migrated namespace. Once triggered the monitor will inform the migrator a namespace change is pending. Operations of interest include VOP_RENAME, VOP_RMDIR, and VOP_REMOVE.

Several of these operations will not actually block the operation but will be monitored in order for the file system monitor to provide clues to future access so a migration can be scheduled in advance of the user access of the data thereby preventing the need to block file system operations.

In all cases the migrator threads that are responsible for data movement will have a different path through these monitors that will allow normal file system access in order to prevent the kernel component from interfering with data movement.

7.2 Interfaces

7.2.1 User-visible

There are no user-visible interfaces exposed by the file system monitor.

7.2.2 Internal

There are two interfaces which will be used, one configuration and control of the monitors and another to send asynchronous file system events.

For configuration and control, an ioctl interface will be used to start and stop monitors as well as some minor ancillary functions (e.g. print out existing migration tasks/monitors)

For asynchronous event notification, two doors interfaces to user-space will be used, one for file requests and the other for directory requests. An event data structure representing file system activities will be sent up through the door into the migrator. Once migration is complete, another data structure will be sent down containing status and attributes to be used to satisfy future VOP_GETATTR calls on the target entity.

7.3 Operati on

The file system monitor will be a loadable module that will be installed at system boot time. Until a migration task is started and a monitor is installed, the module will be dormant.

When requested, the file system monitor will install a monitor on a vnode. Asynchronously, as file system operations trip the monitor, requests will be sent to user space to trigger migration events. Completion of the migration of the target directory/file will then allow the file system operation to complete to disk.

Upon failure of the user space component in which the file system monitor does not reach user space component, a retry process will be initiated. When the retries have been exhausted an error will be returned in response to the file system operation.

The following directory operations will trigger the specified behavior on directories that have not yet completed namespace migration:

Operation	Behavior
VOP_LOOKUP	Block operation, request namespace migration
VOP_OPEN	Block operation, request namespace migration
VOP_CREATE	Block operation, request namespace migration
VOP_GETATTR	Substitute select source attributes into return structure, register interest.
VOP_READDIR	Block operation, request namespace migration

The following directory operations will trigger the specified behavior on directories that have completed namespace migration:

Operation	Behavior
VOP_RENAME	Block operation, notify migrator of namespace change
VOP_REMOVE	Block operation, notify migrator of namespace change
VOP_RMDIR	Block operation, notify migrator of namespace change

The following file/entity operations will trigger the specified behavior on entities that have completed namespace migration, but no data migration:

Operation	Behavior
VOP_OPEN	Register interest, allow operation to complete
VOP_GETATTR	Substitute select source attributes into return structure, register interest.
VOP_READ	Block operation, request migration
VOP_WRITE	Block operation, request migration

Again, migrator threads will be given normal file system access to prevent the monitors from getting in the way of migration activities.

8 DMM Database Architecture

DMM will utilize a file system-backed database to store configuration and to track migration state. Utilization of a database will be opaque to the user.

The database will be accessible through a discrete component which provides an API intended to both hide database implementation detail and to encapsulate as much database-related functionality as possible, with the intention of removing this concern from the application.

8.1 Descri ption

The database component serves two main purposes: it provides persistent storage for application configuration, and it provides a repository for the progress and state of each migration. Architecturally, each purpose corresponds to a separate database type: a configuration database and a migration database. While the configuration database is singular in nature, a separate migration database is maintained for each migration.

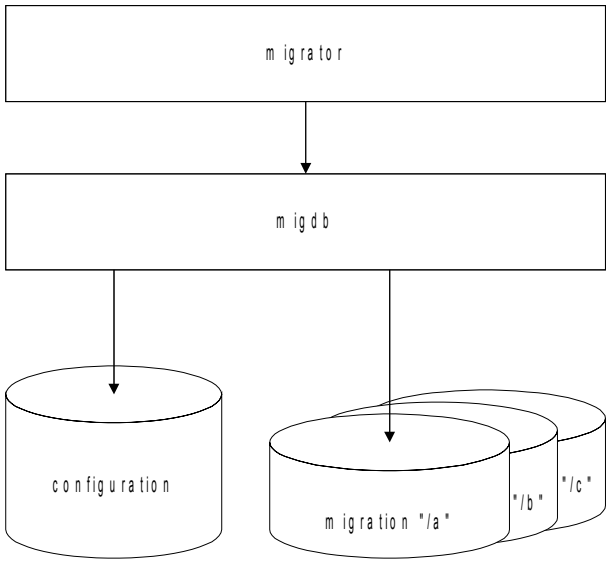


Illustration 3: Database Overview

The database component has two basic conceptual layers: the API layer and the engine layer. The API layer provides an application-tailored interface, using native data structures and types. Translation between native data and engine-specific data is handled at the API layer.

The engine layer interfaces with a specific database engine implementation. MySQL will be utilized as the database engine, although conceivably any engine implementation with a basic set of database operational capabilities could be used; the specifics of the engine are opaque to the application.

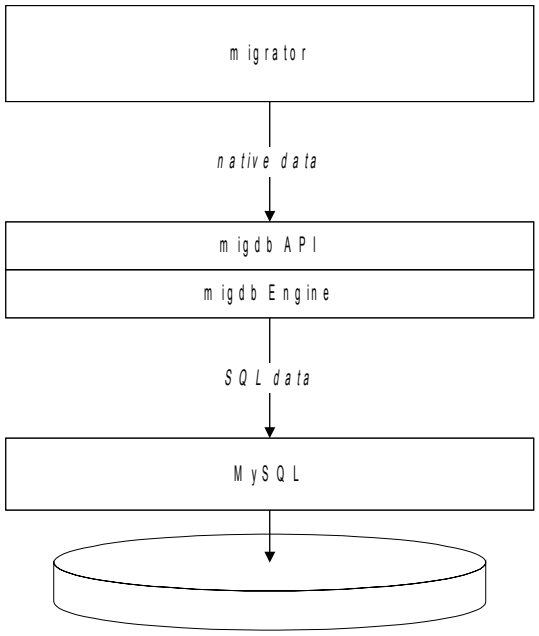


Illustration 4: Database Engine Overview

The relational capabilities of a SQL-based engine allows much of the basic fetch and update functionality to be handled by the engine itself, rather than the API layer.

8.2 Data Structure

8.2.1 API

```
typedef struct
{
      TBD
}
migdb_opts_t;

Additional data TBD.
```

8.2.2 Schema

8.2.2.1 Configuration Databse

TBD

8.2.2.2 Migration Database

TBD

8.3 Interfaces

8.3.1 External

```
int db_init(migdb_opts_t *opts);
```

Initializes the database.

Parameters:
opts: Database options.

Return Values:
0 on success, specific non-zero error code on failure.

```
void db_term(void);
```

Terminates the database. This is one-time call, to be used when the application shuts down.

Parameters:
N/A

Return Values:
N/A

Additional API functions TBD

8.3.2 Internal

TBD

8.4 Operati on

The database component is structured as a library. The API is synchronous and passive; no processes or threads are initialized within the component.