PVFS2 Distribution Design Notes

PVFS Development Team

May 2004

1 Introduction

This document is intended to serve as a reference for the design of the PVFS2 file distributions. This should (eventually) include a description of the mechanism and a guide on developing new distribution methods.

Distributions in PVFS are a mapping from a logical sequence of bytes to a physical sequence of bytes on each of several I/O servers. To be of use to PVFS system code this mapping is expressed as a set of methods.

Files in PVFS appear as a linear sequence of bytes. A specific byte in a file is identified by its offset from the start of this sequence. This is refered to here as a *logical offset*. A contiguous sequence of bytes can be specified with a logical offset and an extent.

Requests for access to file data can be to PVFS servers using various request formats. Regardless of the format, the same data request is sent to all PVFS servers that store part of the requested data. These formats must be decoded to produce a series of contiguous sequences of bytes each with a logical offest and extent.

PVFS servers store some part of the logical byte sequence of each file in a linear sequence of bytes or byte stream within a data space associated with the file. Bytes within this byte stream are identified by their offset from the start of the byte stream referred to here as a *physical offset*. On the server the PVFS distribution methods are used to determine which portion of the requested data is stored on the server, and where in the associated byte stream the data is stored.

2 System Interface Distributions

PVFS2 users should be able to utilize distributions effectively through the system interface. API's are exposed that allow users to create files with the user-specified distribution. In the case that no distribution is specified (i.e. the NULL distribution is specified), the default distribution, simple stripe is used. The system interface must be initialized before distributions may be accessed.

The external distribution API is exposed to users via the following data types and functions:

```
struct PVFS_sys_dist;
```

The system interface distribution structure. It contains the distribution identifier (i.e. the name) and a pointer to an instance of the distribution parameters for this type distribution. In general, the user should not modify the data within this struct.

```
int PVFS_sys_create( char* entry_name,
                     PVFS_object_ref ref,
                     PVFS_sys_attr,
                     PVFS_credentials credentials,
                     PVFS_sys_dist* dist,
                     PVFS_sysresp_create* resp );
```

Creates a file using the specified distribution. If no distribution is specified, the default distribution *simple_stripe* is used during creation. The distribution used during file creation is stored with the file and may not be changed later. Altering the distribution used to store the file contents could result in data corruption.

```
PVFS_sys_dist* PVFS_sys_dist_lookup( const char* name );
```

Allocates a new distribution instance by copying the internal distribution registered for the supplied name. Note that the internal distribution has additional data not exposed thru the system interface, but that should be fully configurable thru the distribution parameters.

```
int PVFS_sys_dist_free( PVFS_sys_dist* dist );
```

Deallocate all system interface resources allocated during distribution lookup.

```
int PVFS_sys_dist_setparam( PVFS_sys_dist* dist,
                            const char* param,
                            void* value );
```

Set the distribution parameter specified by the string *param* to *value*. The strings used to specify parameters are distribution defined but should generally correspond to the field name in the distributions parameter struct. All parameters must be set before the distribution is used in file creation. Once a file is created, there is no safe way to modify the distribution parameters for that file.

3 Distribution Initialization

All distributions are registered during PVFS2 initialization. Although there has been some discussion about having distributions function as loadable modules, there is currently no support for that feature within PVFS2.

All available distributions are loaded into a registration table during initialization and registered with the distribution name as the key. When a user then wishes to create a distribution later, a lookup can be performed with the distribution name, and a copy of the registered distribution is returned. The registered distribution itself is never actually modified after registration. The only opportunity to modify the registered distribution is during the registration itself. Each distribution implements a callback method named *register_init* that is called during registration. The function signature is described completely below, for now we merely want to note that this function is called exactly once (at registration time), and it is generally used by distributions to setup the distribution parameter strings (for use in PVFS_sys_dist_setparam), and to set default parameter values.

Distribution initialization is performed by the function PINT_dist_initialize() in pint-dist-utils.h. In order to add a new distribution to the table of registered distributions, it will be neccesary to modify this function.

4 Internal Distribution Representation

PVFS2 distributions are internally represented with the struct PINT_dist. This structure contains a pointer to the distribution name, methods, parameters and various sizes. The internal distributions are used on both the clients and the metadata server, as well as being stored physically with the file metadata.

When a user creates a file, the system distribution supplied, or the default distribution is exchanged for a corresponding PINT_dist structure. It is this structure that will be used for any further operations performed on the file and stored in the metadata for the file.

The client and server both use the distribution methods to fulfill the request from the client to the server to locate a specific byte range in a specific file. All this processing is performed within the PINT request for the file and byte range. The main difference in the client and server processing is the way segments are built is different as they represent the distribution of data from the various servers, not the distribution of data on the server (What in the world does this sentence mean?!?)

Distribution parameters are defined in the exported header for the distribution (e.g. for the simple stripe distribution, the header file is pvfs2-dist-simple-stripe.h). The distribution methods are usually defined in a corresponding implementation file in the io/description subsystem (e.g. the simple stripe implementation is in io/description/dist-simple-stripe.c).

The methods defined for each distribution allow it to completely specify how the file data is mapped to the PVFS2 disk abstraction, the data file object. The one possible exception to this is that distributions cannot currently assert their preference in how data file objects are mapped to data servers. This is planned in the near future, however their is no current consensus on how to improve upon the current round robin mapping approach (see PINT_bucket_get_next_io).

5 Distribution Parameters

The parameters for each distribution are defined in a struct defined specifically for the distribution, and an individual instance of the parameters is stored in the metadata of every file.

Both the PVFS_sys_dist and PINT_dist data structures maintain a pointer to the same distribution parameters. The parameters are passed into every call to distribution code so that distribution can modify its behavior as neccesary. The distribution provider can also provide a method for setting the distribution parameters explicitly as described in the distribution methods below.

6 Distribution Methods

The distribution methods are the individual code used by each distribution to perform mappings between the logical file data and the data file objects. The methods also provide a mechanism for encoding/decoding the distribution parameters, determining the number of data file objects to create for a file, modifying distribution parameters, and distribution registration tasks. For some of the methods a default implementation is available that may be acceptable for most distributions.

```
PVFS_offset logical_to_physical_offset( void* params,
                                        uint32_t dfile_nr,
                                        uint32_t dfile_ct,
                                        PVFS_offset logical_offset );
```

Given a logical offset, return the physical offset that corresponds to that logical offset. Returns a physical offset. The return value rounds down to the largest physical offset held by the I/O server if the logical offset does not map to a physical offset on that server.

```
PVFS_offset physical_to_logical_offset( void* params,
                                        uint32_t dfile_nr,
                                        uint32_t dfile_ct,
                                        PVFS_offset physical_offset)
```

Given a physical offset, return the logical offset that corresponds to that physical offset. Returns a logical offset. The input value is assumed to be on the current PVFS server.

```
PVFS_offset next_mapped_offset( void* params,
                                uint32_t dfile_nr,
                                uint32_t dfile_ct,
                                PVFS_offset logical_offset)
```

Given a logical offset, find the logical offset greater than or equal to the logical offset that maps to a physical offset on the current PVFS server. Returns a logical offset.

```
PVFS_size contiguous_length( void* params,
                             uint32_t dfile_nr,
                             uint32_t dfile_ct,
                             PVFS_offset physical_offset)
```

Beginning in a given physical location, return the number of contiguous bytes in the physical bytes stream on the current PVFS server that map to contiguous bytes in the logical byte sequence. Returns a length in bytes.

```
int get_num_dfiles( void* params,
                    uint32_t num_servers_requested,
                    uint32_t num_dfiles_requested )
```

Returns the number of data file objects to use for the requested file. The number of servers requested and number of data files requested are hints from the user that the distribution can ignore if neccesary. A default implementation of this function is provided in pint-dist-utils.h that returns the number of servers requested (which is usually the number of data servers in the system).

```
int set_param( const char* dist_name, void* params
               const char* param_name, void* value )
```

Set the distribution parameter described by *param_name* to *value*. A default implementation is provided in pint-dist-utils.h that can handle parameters that have been previously registered.

```
void encode_lebf( char** pptr, void* params )
```

Write *params* into the data stream pptr in little endian byte format.

```
void decode_lebf( char** pptr, void* params )
```

Read *params* from the data stream pptr in little endian byte format.

```
void registration_init( void* params )
```

Called when the distribution is registered (i.e. once). Used to set default distribution values, register parameters, or any other initialization activity needed by the distribution.