PVFS2 distributions design notes

PVFS Development Team

February 2002

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 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.

The PVFS servers utilize the distribution methods to convert a logical offset and extent into one or more physical offsets and extents relative to the data space on the file server. We next describe the methods used by the PVFS server and provide pseudo code for their use in decoding a request.

2 Methods

```
PVFS_offset logical_to_physical_offset (PVFS_Dist_parm *dparm,
     uint32_t server_nr, uint32_t server_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 (PVFS_Dist_parm *dparm,
      uint32_t server_nr, uint32_t server_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 (PVFS_Dist_parm *dparm,
      uint32_t server_nr, uint32_t server_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 (PVFS_Dist_parm *dparm,
      uint32_t server_nr, uint32_t server_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.

PVFS distribution processing pseudo code:

```
// INPUTS
   PVFS_offset offset;      // logical offset of requested data
   PVFS_size size;          // size of requested data
   int req_type;            // type of read A_READ or A_WRITE
   PVFS_Dist_parm *d_p;     // point to file distribution parameter structure
uint32_t server_nr;    // number of iods data distributed on
uint32_t server_ct;  // ordinal number this iod
PVFS_distribution *dist; // distribution methods

// LOCALS
   PVFS_offset loff;
   PVFS_offset diff;
   PVFS_offset poff;
   PVFS_size   sz;
   PVFS_size   fraglen;
```

```
loff = (*dist->next_mapped_offset) (d_p, server_nr, server_ct, offset);
while ((diff = loff - offset) < size)
{
   poff = (*dist->logical_to_physical_offset)(d_p,server_nr,server_ct,loff);
   sz = size - diff;
   if (poff+sz > m_p->fsize && req_type==A_READ) // check for append
   {
      /* update the file size info */
      if (update_fsize() < 0) return(-1);
      if (poff+sz > m_p->fsize) sz = m_p->fsize - poff; // stop @ EOF
      if (sz <= 0)
      {
         // hit end of file
         return(1);
      }
   }
   fraglen = (*dist->contiguous_length) (d_p, server_nr, server_ct, poff);
   if (sz <= fraglen || m_p->pcount == 1) // all in 1 block
   {
      create_segment (poff, sz);
      return(0);
   }
   else // frag extends beyond this stripe
   {
      create_segment (poff, fraglen);
   }
   /* prepare for next iteration */
   loff  += fraglen;
   size  -= loff - offset;
   offset = loff;
   loff = (*dist->next_mapped_offset) (d_p, server_nr, server_ct, offset);
}
```

3 Client Processing

PVFS clients run the same code as a PVFS server, but 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.

4 Distribution Registration

Distributions are registerd with PVFS byt either compiling a distribution method entry into the distribution table of the PVFS code or by dynamically adding a method entry to the table. Distribution method entries are registration functions are defined as follows:

```
struct PVFS_Distribution {
    char *dist_name;
    int param_size;
    PVFS_offset (*logical_to_physical_offset) (PVFS_Dist_parm *dparm,
         uint32_t server_nr, uint32_t server_ct,
          PVFS_offset logical_offset);
    PVFS_offset (*physical_to_logical_offset) (PVFS_Dist_parm *dparm,
          uint32_t server_nr, uint32_t server_ct,
          PVFS_offset physical_offset);
    PVFS_offset (*next_mapped_offset) (PVFS_Dist_parm *dparm,
          uint32_t server_nr, uint32_t server_ct,
          PVFS_offset logical_offset);
    PVFS_size (*contiguous_length) (PVFS_Dist_parm *dparm,
          uint32_t server_nr, uint32_t server_ct,
          PVFS_offset physical_offset);
};

   void PVFS_register_distribution(struct PVFS_distribution *d_p);

   void PVFS_unregister_distribution(char *dist_name);
```

Dynamically loaded modules are expected to provide initialization and cleanup functions as follows:

```
   void init_module();

   void cleanup_module();
```

The init_module function would generally register the distribution and the cleanup_module function would generally unregister the distribution.

5 Distribution Parameters

Distributions may define a structure containing parameters for the distribution which are assigned on a per-file basis, stored with the file metadata, and provided to each method when it is called. Default parameters are provided with the methods and are used if a NULL pointer to distribution parameters is passed into the

method. The definition of the parameter structures should be provided to user programs via an include file, where the parameters can be initialized and passed in to the system through an interface routine.