А.В.Коновалов

# Параллельный ввод-вывод: большие файлы и реальность

# План

- Новости МВС.

- Общий обзор PVFS.
- Системные вызовы параллельного IO.
- Внутренности.

# Что это даёт

- Большие файлы.

  По-настоящему большие. Терабайтные. 200 ГБ есть прямо сейчас.

- Быстрый ввод-вывод.

  250 MB/сек достигалось на um16.

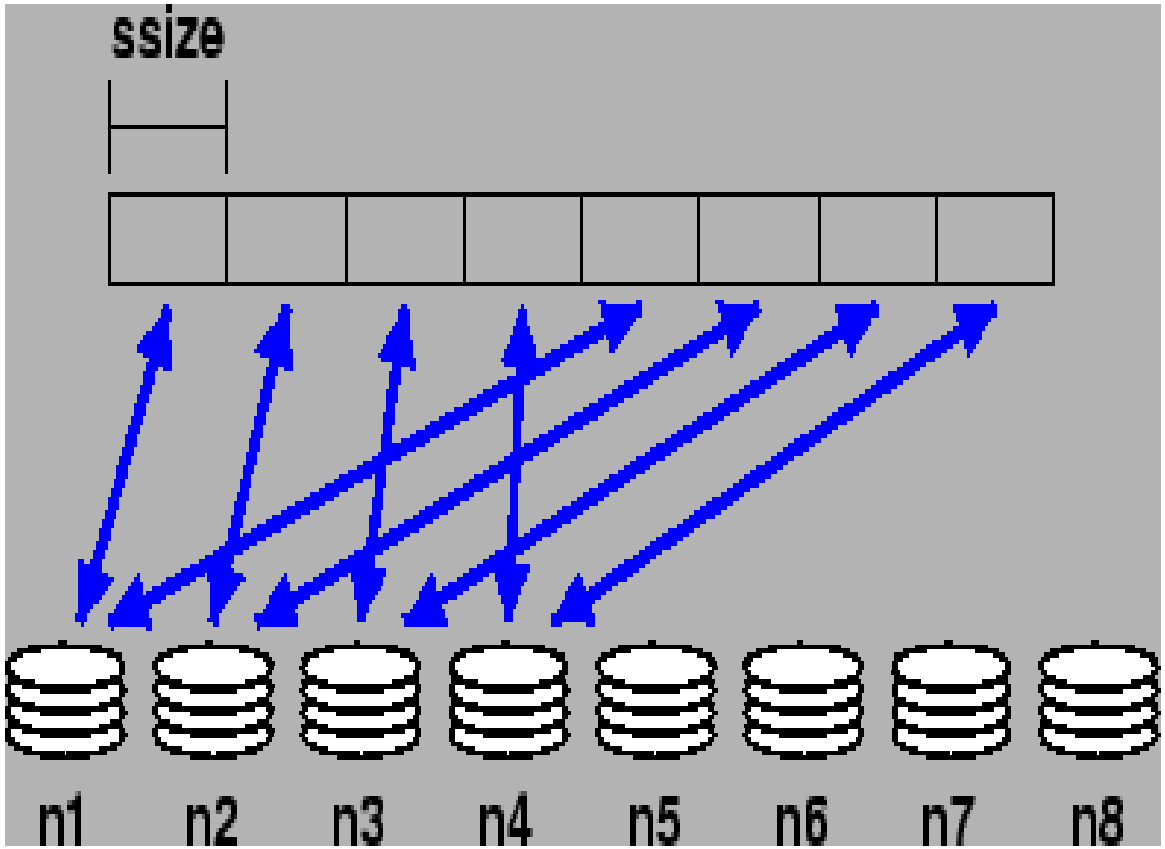
- Параллельно пишем с разных узлов.

  Один узел быстрее 24 MB/s писать не будет.

# Писать с нуля – глупо: PVFS

- Разработана в Clemson University и Аргоннской Национальной Лаборатории (двумя PhD).
- Используется в реальной жизни в Окридже, Ливерморе и пр.
- Субстрат для многочисленных исследовательских и учебных проектов.

# Принцип работы PVFS



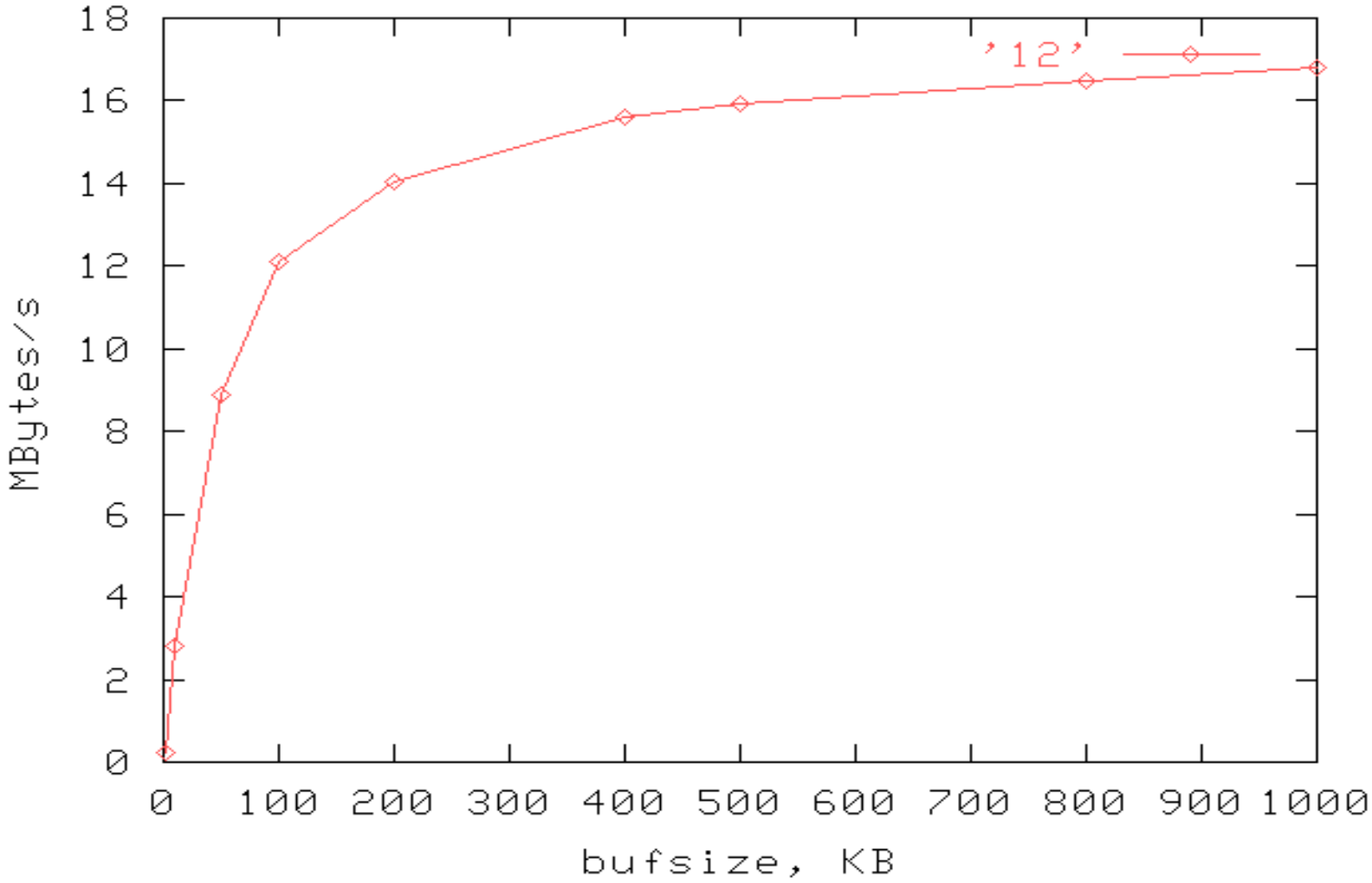
# PVFS – не циркониевый браслет

Не годится для задач:

- С большим числом маленьких обменов.
- mmap “частично есть”, но лучше на него не рассчитывать.
- Активно работающих с метаинформацией.

Если в конце 10-мин. итерации пишется 10 MB, то можно, но бессмысленно (10 MB пишутся секунду).

# Опасность коротких обменов



# 3 семейства системных вызовов

- Обычная работа с файлами. lseek64. long long и integer*8

- MPI-IO. Для Фортрана, C и C++ (загадочное). Подробнее ниже.

- PVFS API. Простой и ненужный прикладным программистам.

# MPI-IO: пример

```
MPI_File fh;

MPI_File_open(MPI_COMM_SELF, “foo.data”,
    MPI_MODE_CREATE | MPI_MODE_RDWR,
    /* советы: как б. работать, сколько узлов etc */
     MPI_INFO_NULL, &fh);
MPI_File_seek(fh, seek_position,MPI_SEEK_SET);

MPI_File_write(fh, buf, nchars, MPI_CHAR,
                        &status);

MPI_File_close(&fh);
```

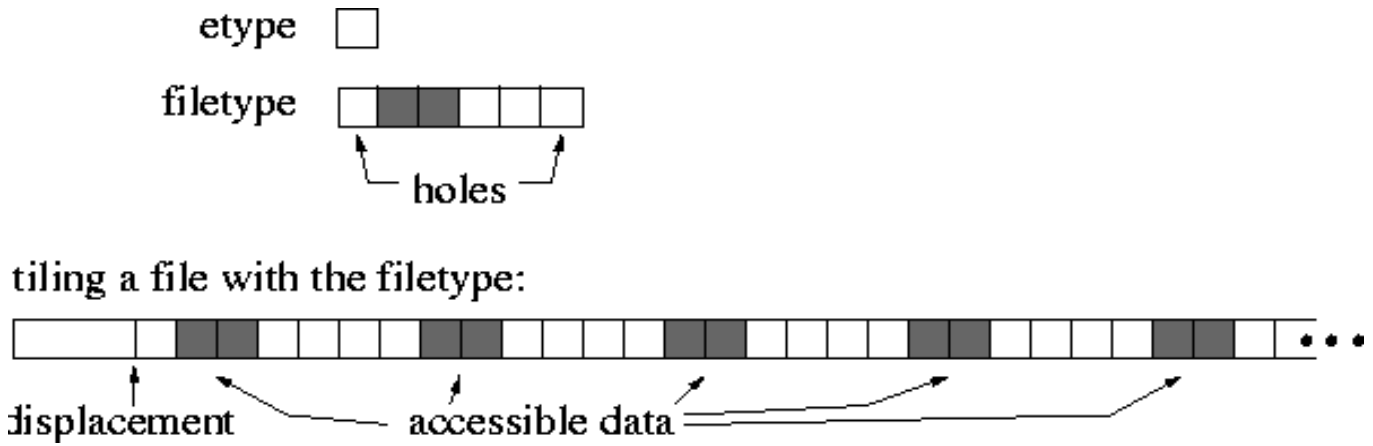
# MPI-IO: основное – это view

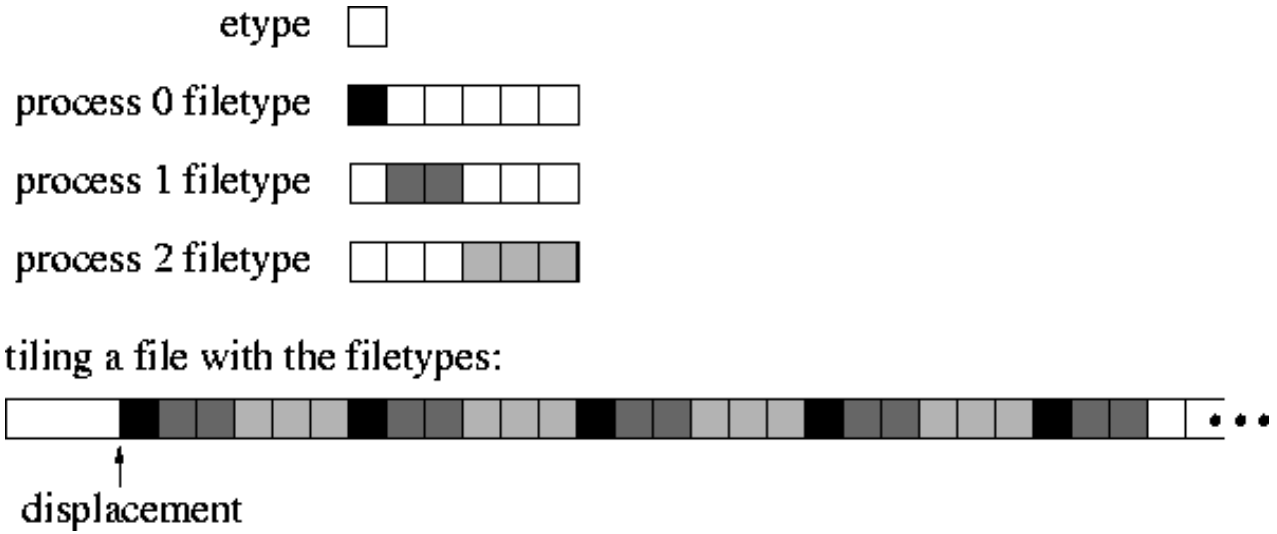
Статическое задание модели доступа на основе проекции логич. структуры на физич. байтики (view).

```
MPI_File_set_view(MPI_File fh,
MPI_Offset disp, // начало (в байтах)
MPI_Datatype etype, // элементарный тип данных
MPI_Datatype filetype, /* Как по процессам
                          ляжет. М.б. с дырками */
char *datarep, // представление на диске
MPI_Info info)
```

Последующие read/write видят только проекцию.

# Применение View



# Тип с дырочками

MPI_Type_vector(int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)

Чётные элементы вектора double m[50];

```
MPI_Datatype even;
MPI_Type_vector(25, 1, 1, MPI_DOUBLE, &even);
```

Это простой пример. Можно прямоугольную матрицу по столбцам нарезать.

# Чтение-запись. Префикс MPI_File_

| смещение | синхр. | индивид. | коллективный |
|---|---|---|---|
| явное | блок. | READ_AT | READ_AT_ALL |
| | неблок. | IREAD_AT | READ_AT_ALL_BEGIN |
| | | | READ_AT_ALL_END |
| на каждом - | блок. | READ | READ_ALL |
| своё | неблок. | IREAD | READ_ALL_BEGIN |
| | | | READ_ALL_END |
| общее | блок. | READ_SHARED | READ_ORDERED |
| | неблок. | IREAD_SHARED | READ_ORDERED_BEGIN |
| | | | READ_ORDERED_END |

# Пример коллективного чтения

MPI_File_read_ordered(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)
читаем count элементов типа datatype в buf
Пусть в файле лежат double, а тек. позиция – 10.
Тогда
double m[2];
MPI_File_read_ordered(f, m, 2, MPI_DOUBLE, &stat);
поместит в m в 0-м процессе коммуникатора, в кот. открыли f 10 и 11 эл-т файла,
на 1-м – 12 и 13, на 2-м – 14 и 15 и т.д.

# Реализация

PVFS – это такая ОС Router среди ФС.

Плюсы:

- Это не “концепция”, это на самом деле работает.

Минусы:

- Коллективное использование.

# Выполненные модификации

- Хранение файла на несплошных наборах узлов ввода-вывода.
- Устойчивость к перевызовам узлов даже непосредственно во время записи.
- Резервные узлы ввода-вывода. Миграция.