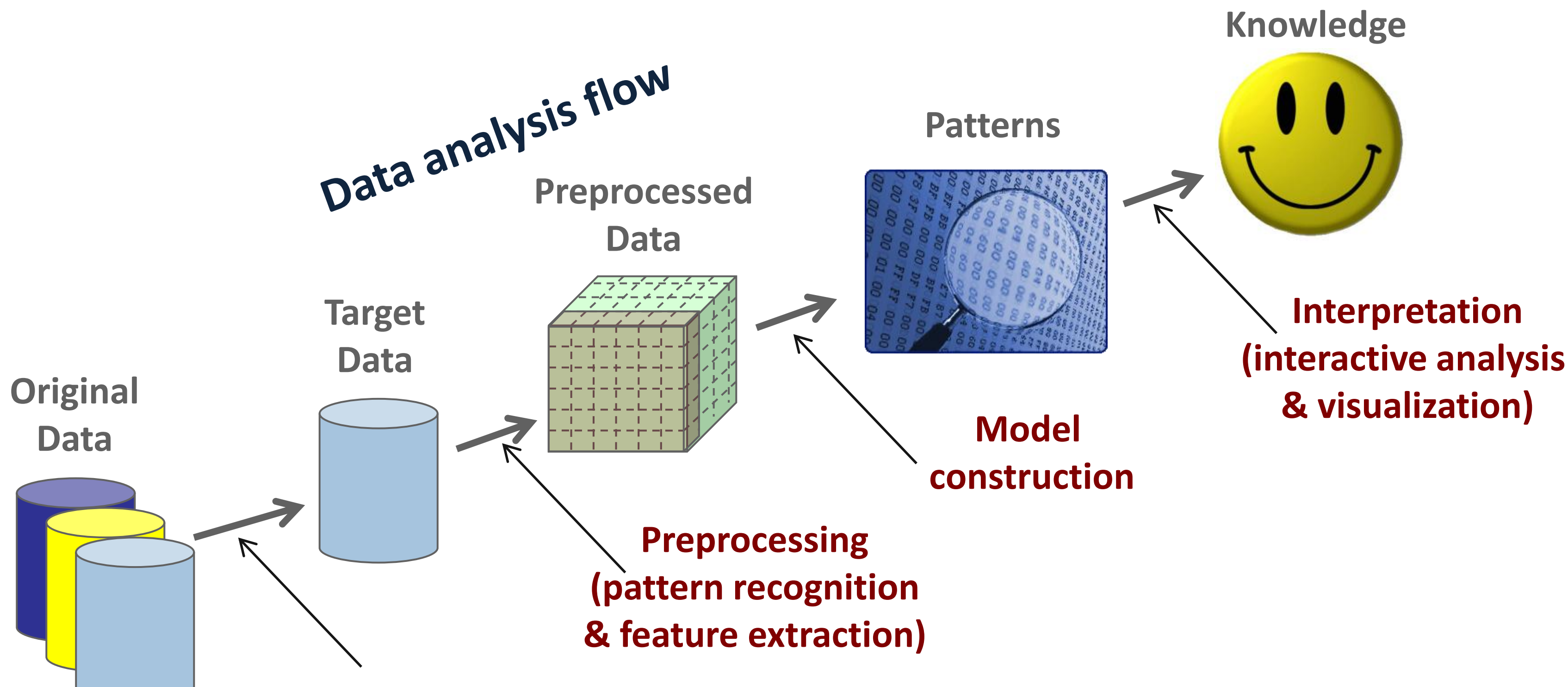
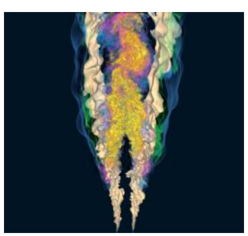
Enabling Active Storage on Parallel I/O Software Stacks

Seung Woo Son
sson@mcs.anl.gov
Mathematics and Computer Science Division

MSST 2010, Incline Village, NV
May 7, 2010

Performing analysis on large data sets is often frustrating





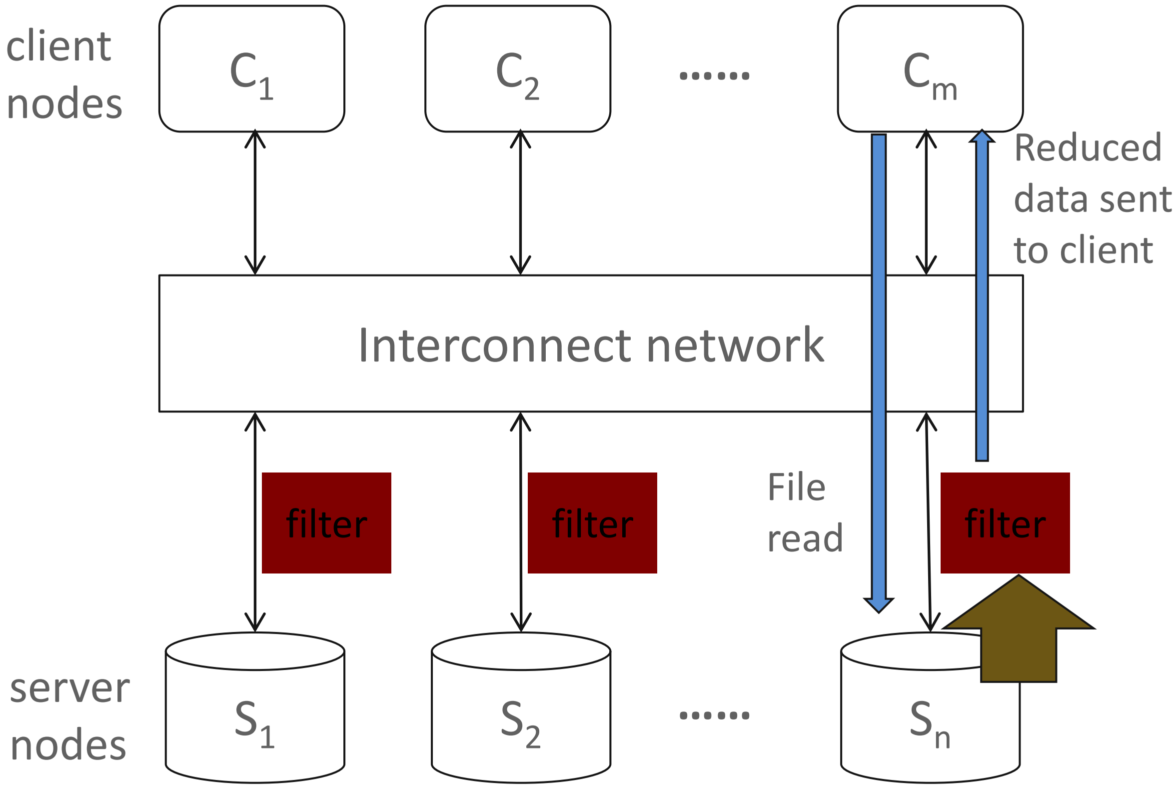
S3D simulations for combustion research are producing 30–130 TB of data per simulation

Scientists and engineers spent too much time on data manipulation, especially moving and reorganizing data

Talk outline

- ✔ Motivation
- Active storage in parallel file systems
- Our prototype
  - Enhanced runtime interface that uses embedded analysis kernels
  - Runtime stripe alignment
  - Server-to-server communication for reduction and aggregation
- Experimental evaluation
- Conclusion

Active storage in parallel file systems



Active storage is a technique for performing data transformations in the storage system

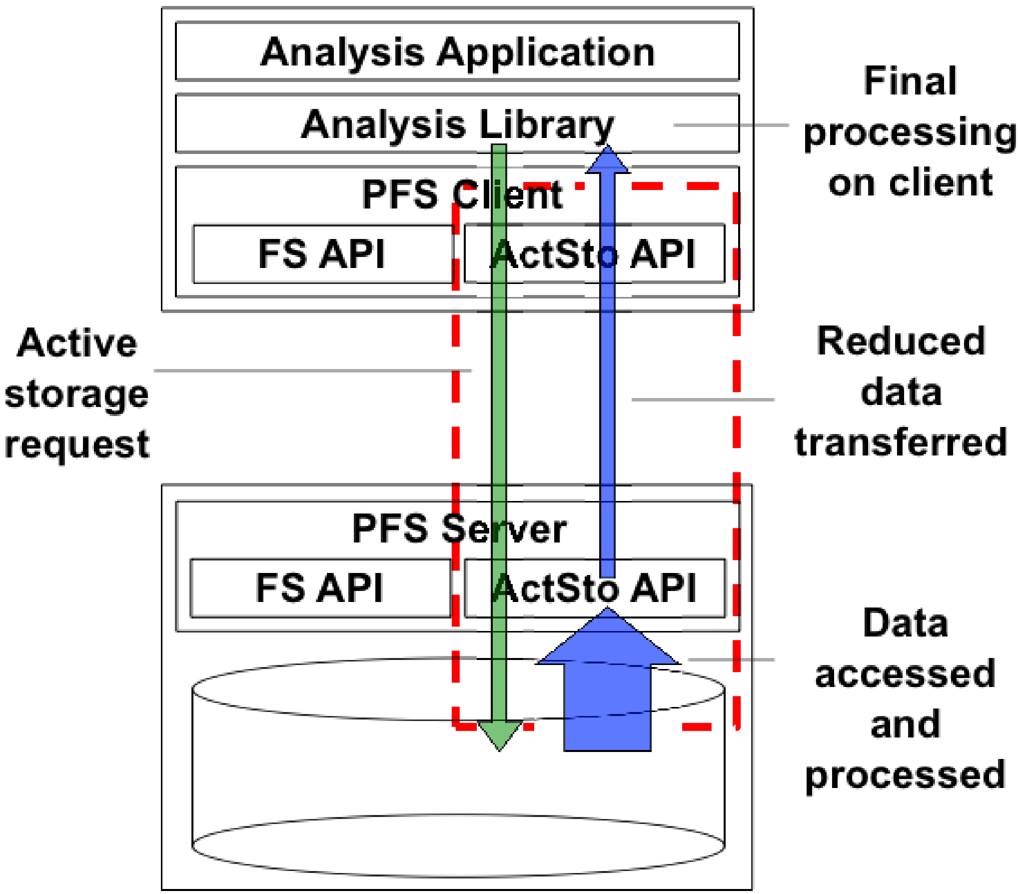
Library or user space implementation; not well integrated into I/O software stacks

Targeting applications that manipulates fundamentally-independent data sets

Lack of reduction and aggregation on the storage nodes

E. Riedel et al., Active disks for large-scale data processing, IEEE Computer, 2001.
J. Piernas et al., Evaluation of active storage strategies for the Lustre parallel file systems, in SC, 2007.

We enable active storage on parallel I/O software stack



1. Enhanced runtime interface (API) to enable active storage operations

2. Runtime data stripe alignment

3. Server-to-server communication primitives for complex analysis

Enhanced runtime I/O interface to trigger embedded analysis kernels

Conventional MPI-based

```
sum = 0.0;
MPI_File_open(&fh);
double *tmp = (double*)malloc(nitem*sizeof(doble));
offset = rank * nitem * type_size;
MPI_File_read_at(fh, offset, tmp, nitem, MPI_DOUBLE, &status)

for(i=0; i<nitem; i++)
    sum += tmp[i]
```

```
...
MPI_File_open(&fh);
...
MPI_File_read_ex(…, SUM, …)
...
```

<client>

```
for(i=0; i<nitem; i++)
    sum += tmp[i];
```

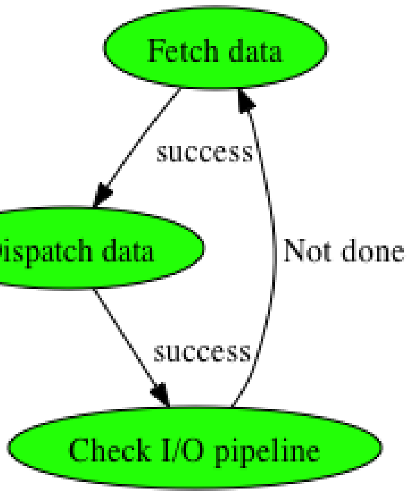
<server>

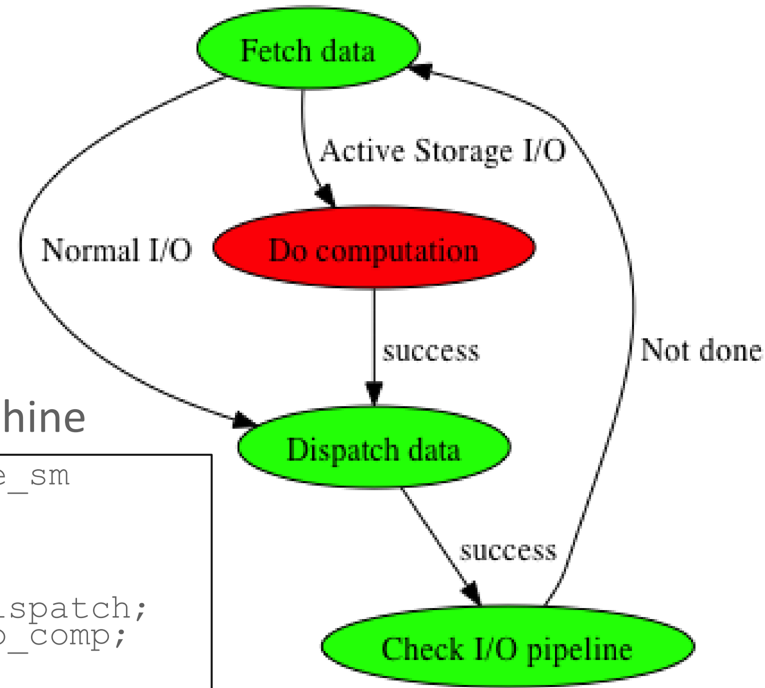
Active storage based

Why MPI?

- MPI is a widely used interface
  - There are a large number of applications
  - Therefore, it might be relatively easy to migrate
- MPI specification provides interfaces where user functions can be embedded into it
  - Enabling the incorporation of data mining and statistical functions easily
- Hint mechanism
  - Passing kernel specific argument to the server, e.g., data types

Mapping embedded analysis kernels into I/O pipeline





pvfs state machine

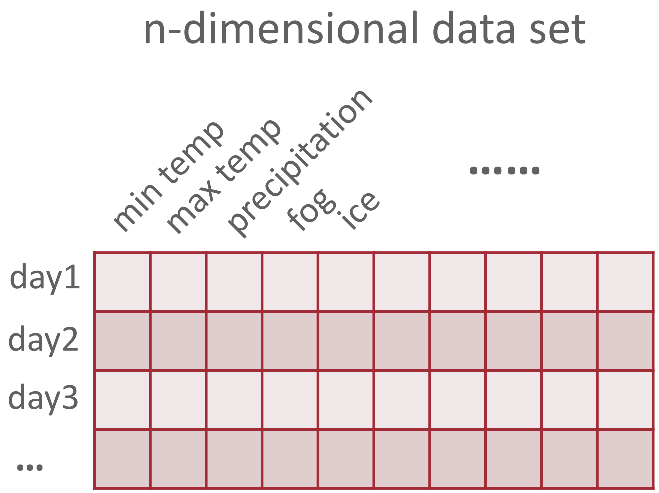
```
machine pvfs_pipeline_sm
{
    state fetch
    {
       run fetch_data;
       normal_op => dispatch;
       active_op => do_comp;

    }
    state do_comp
    {
       run do_comp_op;
       success => dispatch;
    }
    state dispatch
    {
       run dispatch_data;
       success => check_done;
    }
    state check_done
    {
       run check_done_action;
       not_done => fetch;
       default => terminate;
    }
}
```

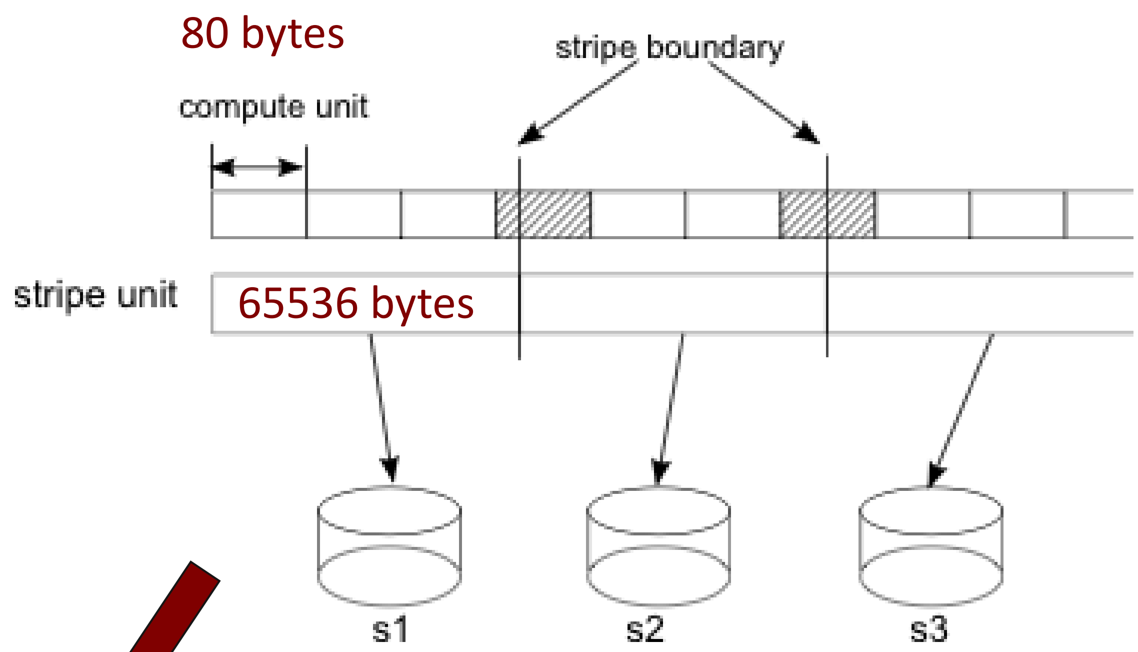
```
static int fetch_data
{
   …
   disk I/O;
   …
}
```

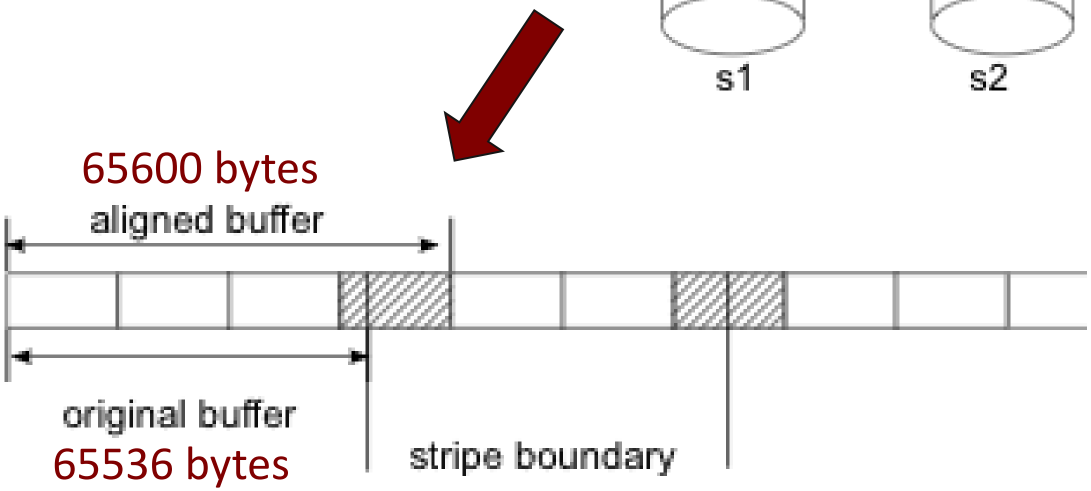
```
static int dispatch_data
{
   …
   send the data;
   …
}
```

```
static int do_comp_op
{
   …
   for(i=0;i<nitem;i++)
      sum += tmp[i] ;
   …
}
```

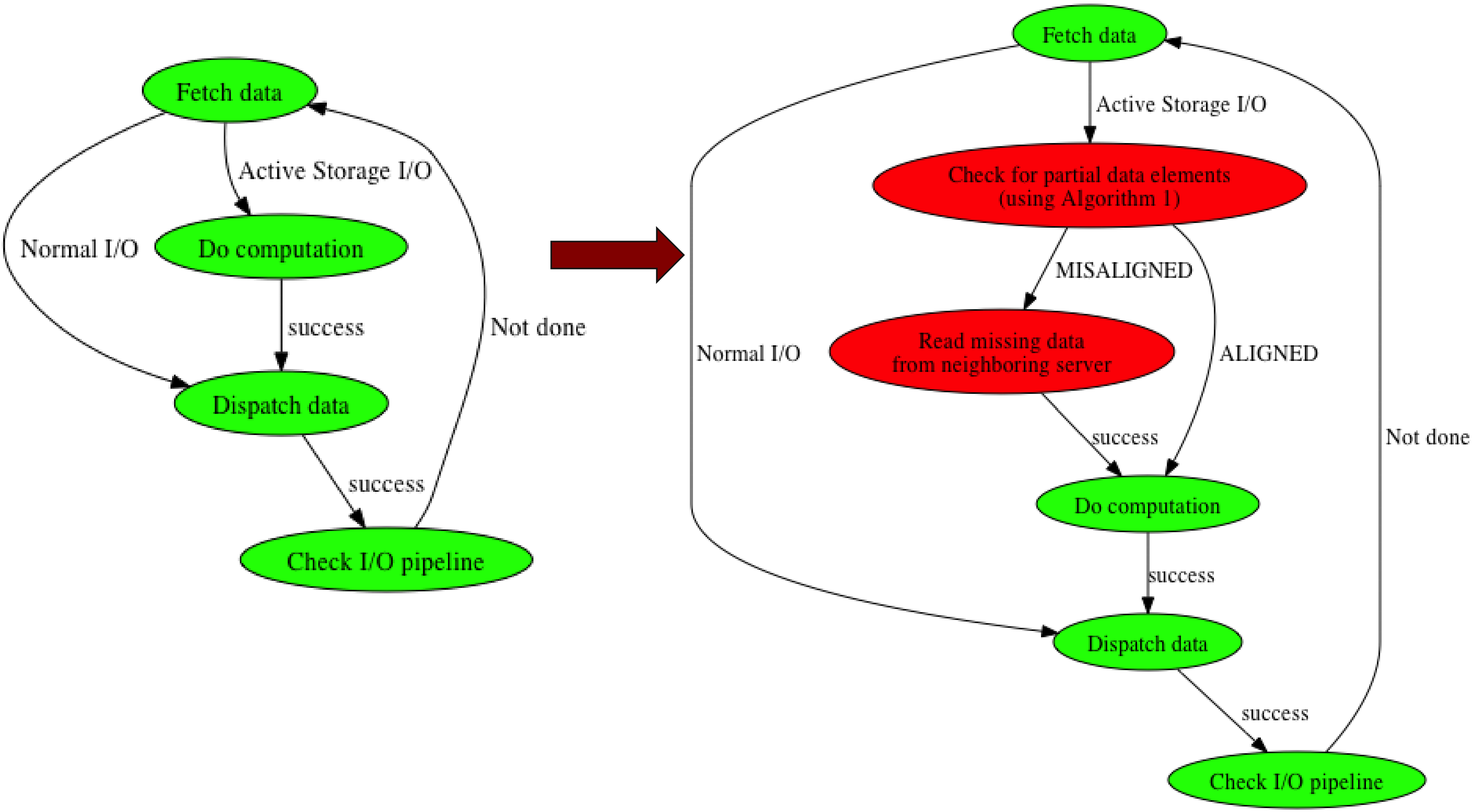
Computational unit is often not perfectly aligned to file stripe unit



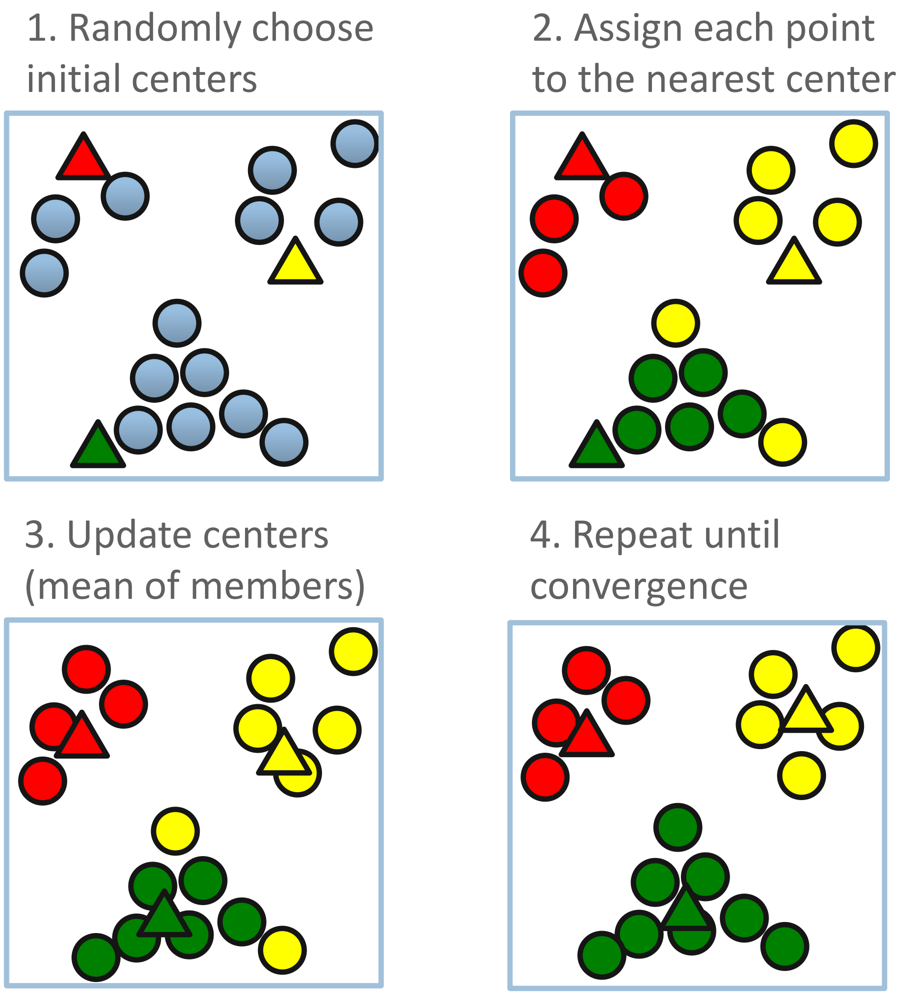




I/O pipeline with data alignment



Server-to-server communication for reduction and aggregation

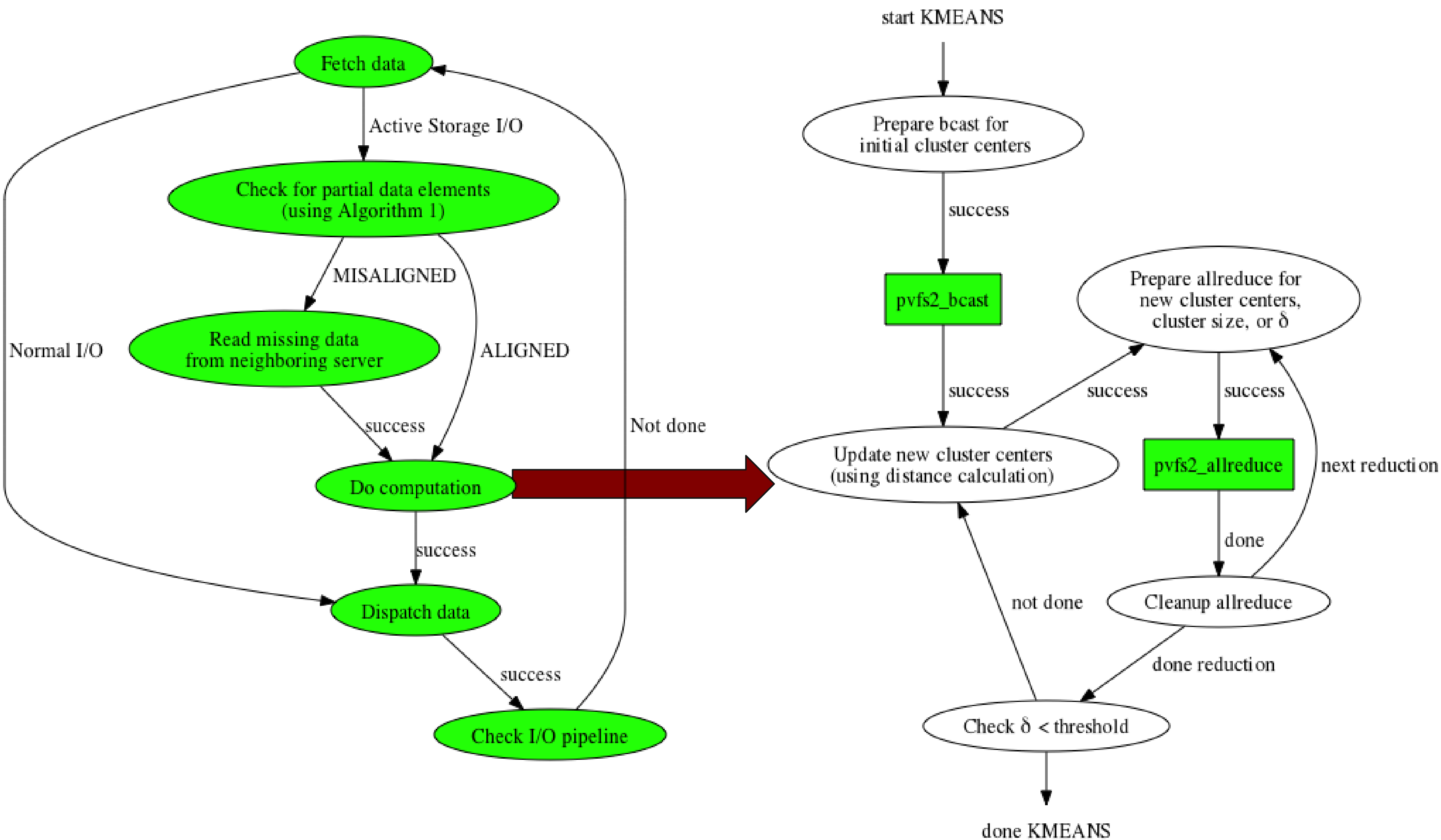


K-means cluster algorithm

Reduction and aggregation can be done on client side (e.g., simple statistical operations)

Complex analysis kernels (e.g., k-means clustering) requires broadcast and reduction during iterative execution

K-means clustering is performed purely on the server side!

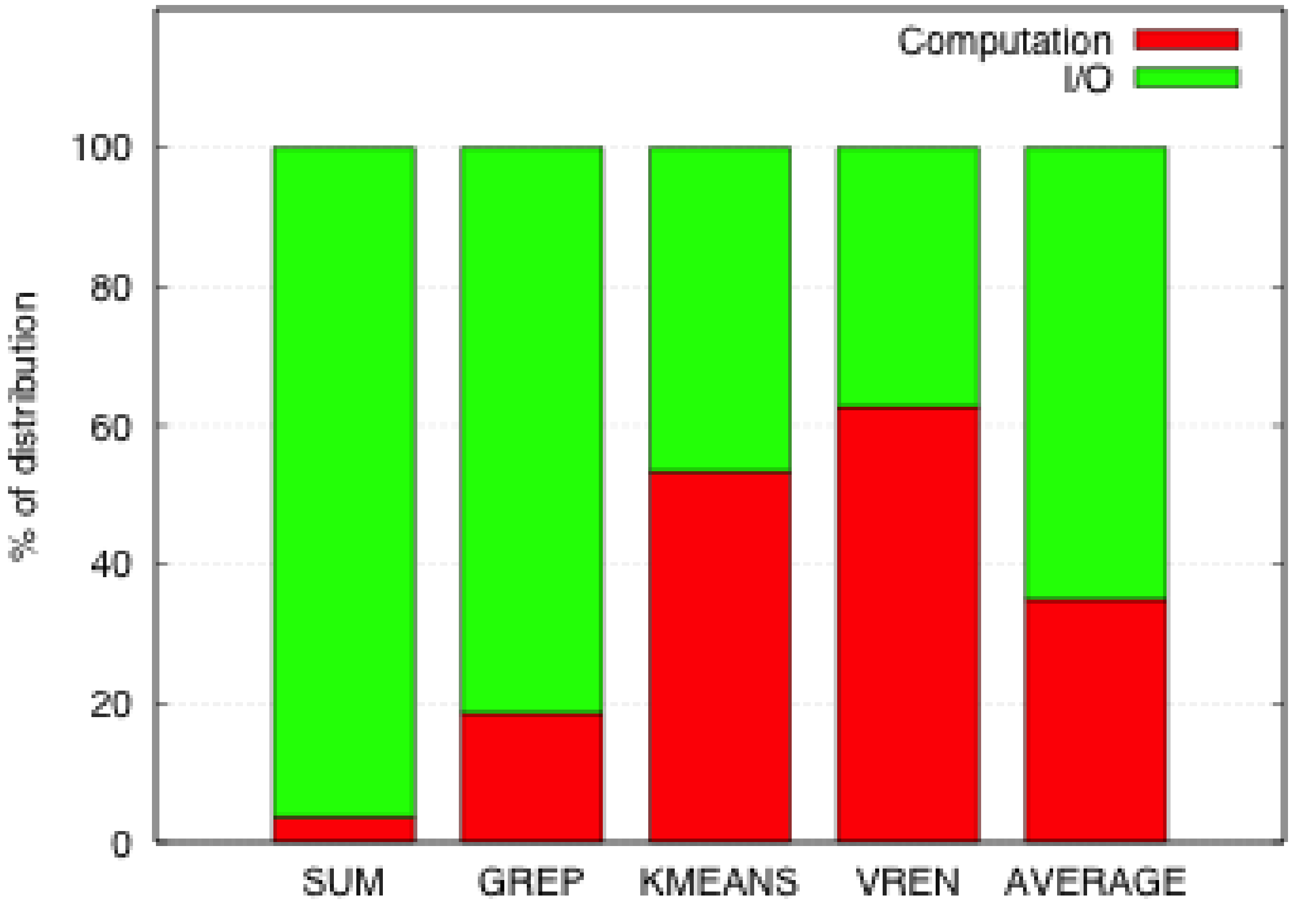


Benchmarks and evaluation platform

Name	description	Base (sec)	Input data	% of filtering
sum	Global reduction	1.38	512 MB	~100%
grep	String pattern matching	1.49	512 MB (4M of 128 string)	~100%
kmeans	K-means clustering algorithm	0.44	40 MB (1M*10 dim of double)	90%
vren	Parallel volume rendering	2.61	103MB (300*300*300 of float)	97%

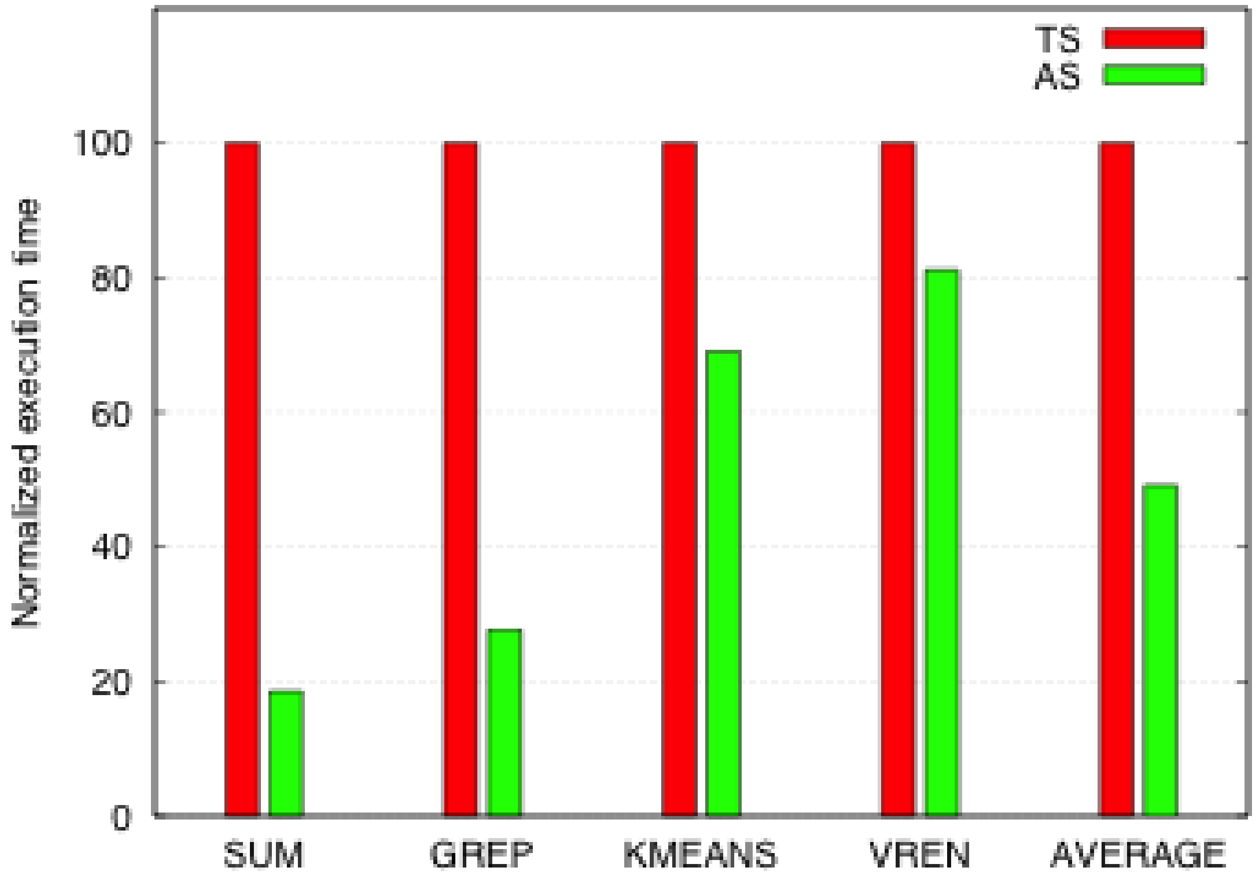
Test cluster	
32 nodes	Dual Intel Xeon Quad Core 2.66 MHz
Main memory	16GB
Storage capacity	~200GB per node
Interconnection network	1 Gb Ethernet
GPU accelerator	2 NVIDIA C1060 GPU card

All benchmarks are I/O dominant



64.4% time is spent on I/O
Benchmarks are executed using 4 nodes

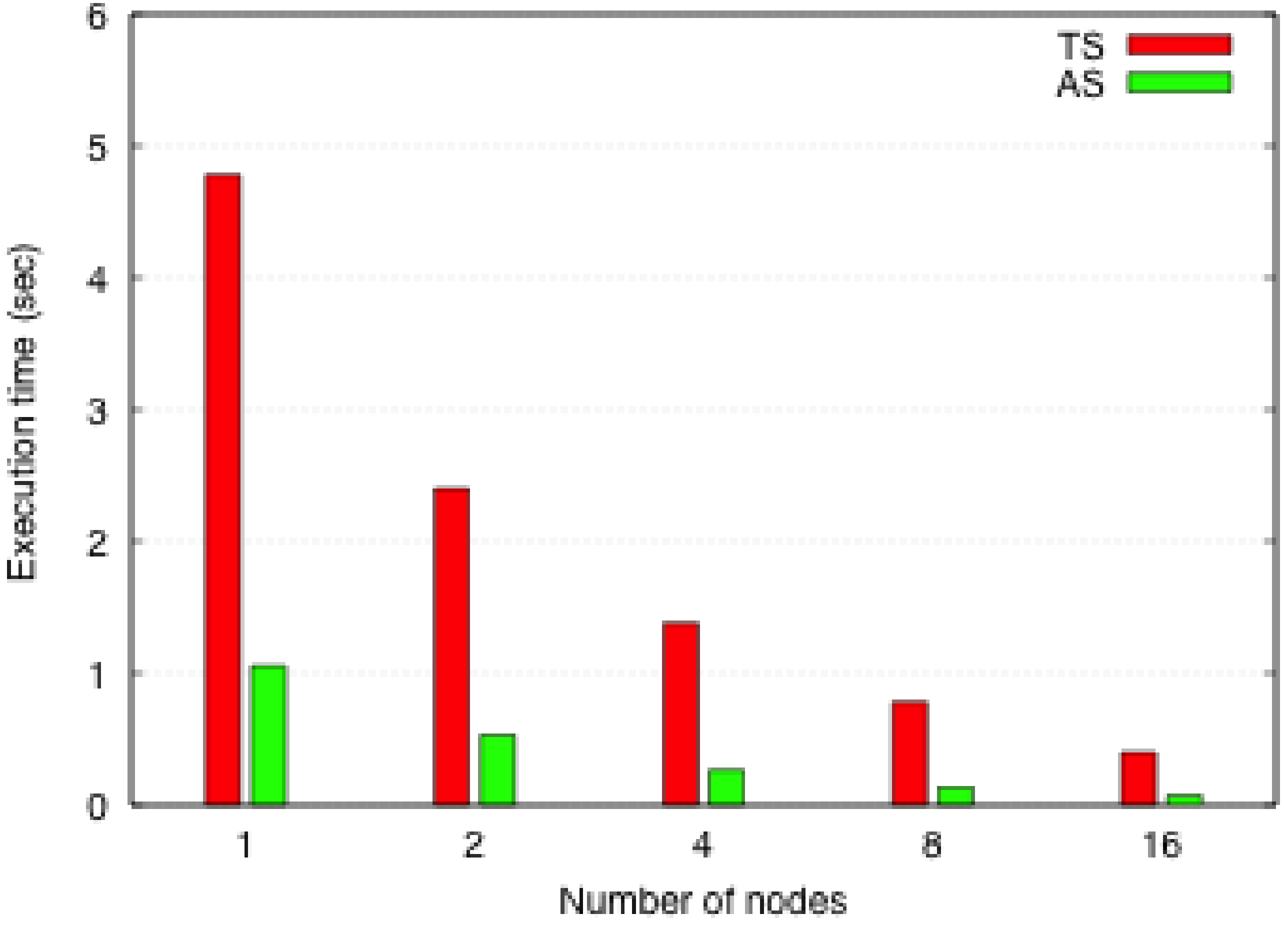
Moving computation to storage server (AS) improves performance significantly



TS: Traditional Storage, 4 client nodes and 4 server nodes
AS: Active Storage, 4 server nodes

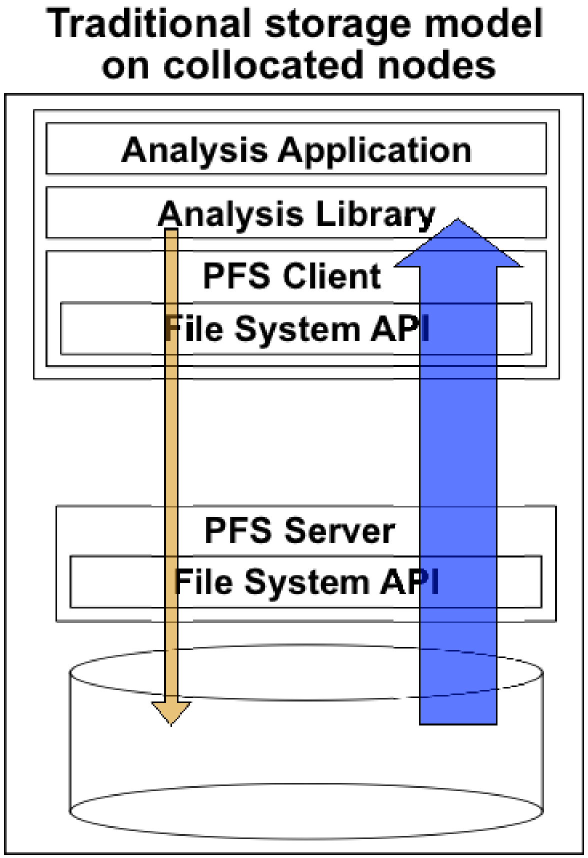
Our approach is scalable w.r.t the different number of nodes

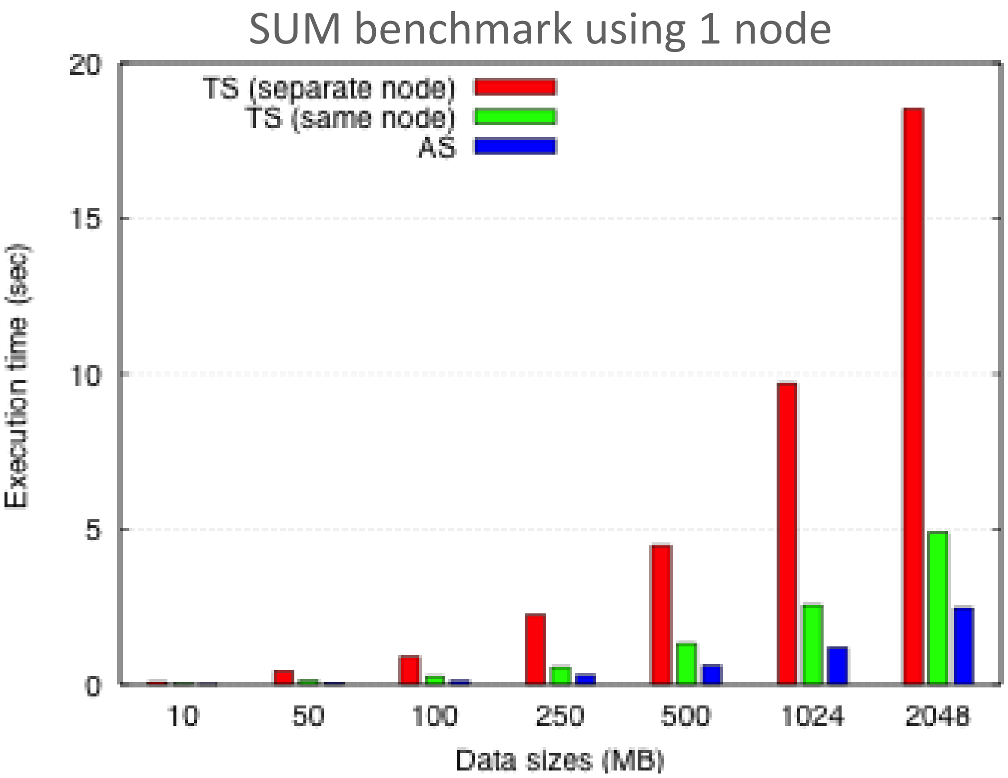
SUM benchmark



Fixed data size: 512MB

Putting client and server together

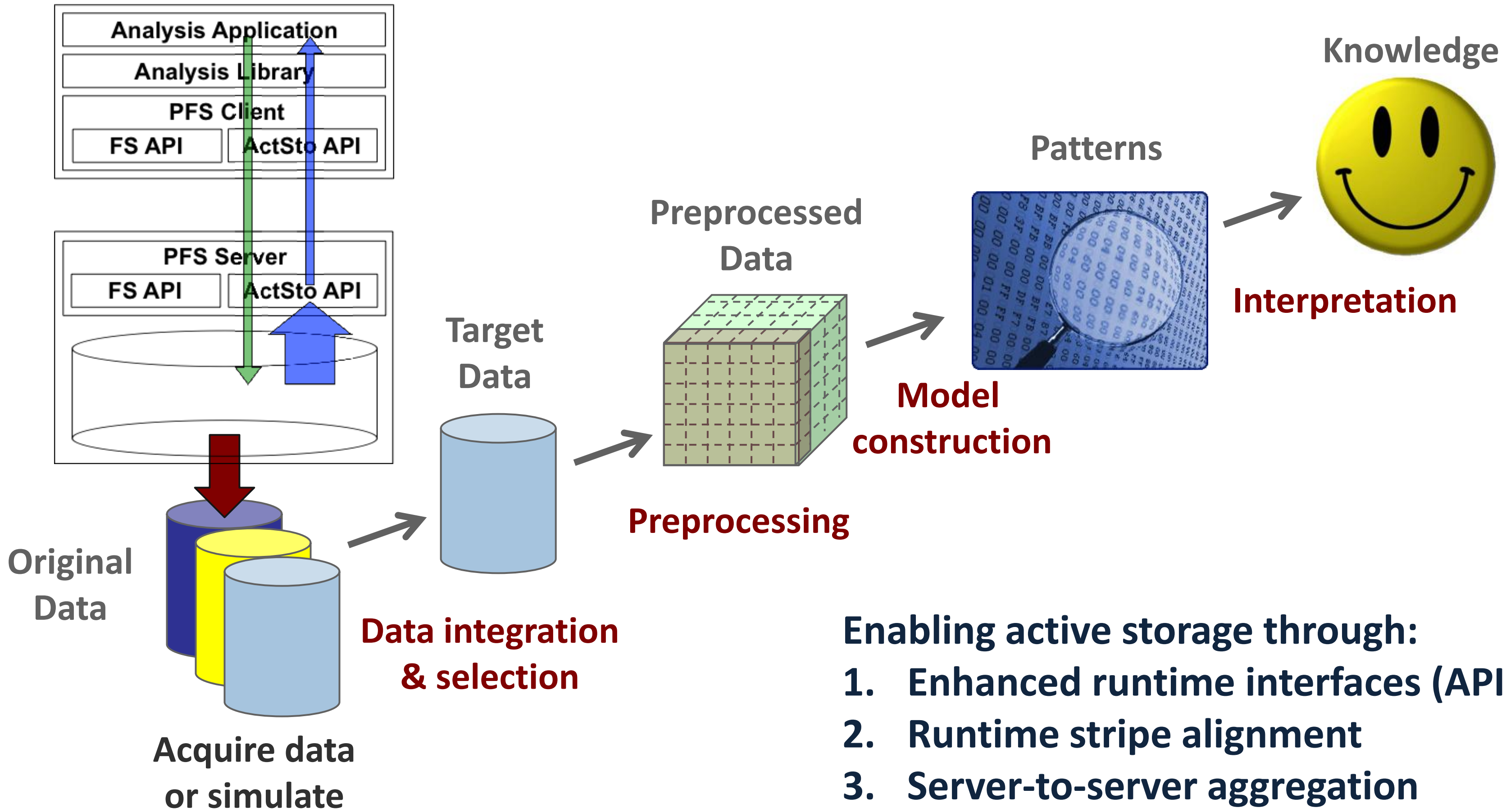




No Inter-node communication,
but Inter-process communication still exists

To achieve this in reality, client should be aware of storage layout!

Conclusion



Enabling active storage through:

1. **Enhanced runtime interfaces (APIs)**
2. **Runtime stripe alignment**
3. **Server-to-server aggregation**

Enabling **Active storage** within parallel I/O software stack removes not only inter-node data transfer, but also inter-process data communication, resulting in a huge performance improvement for data-intensive analysis applications

Acknowledgments

- Department of Energy for funding this work
- Phil Carns, Sam Lang, Rob Ross, Rajeev Thakur (ANL)
- Alok Choudhary, Prabhat Kumar, Wei-Keng Liao, Berkin Ozisikyilmaz (NWU)

Thanks!

Future work

- Function shipping
  - More flexible hint mechanism
- Hadoop style execution
  - Write output result to the local storage
- Scalability analysis
  - NCSA Lincoln cluster: 192 compute nodes and 96 NVIDIA Tesla S1070 accelerator units.
- More benchmarks/applications
  - Visualization and Bioinformatics

Give hints to file servers for more information

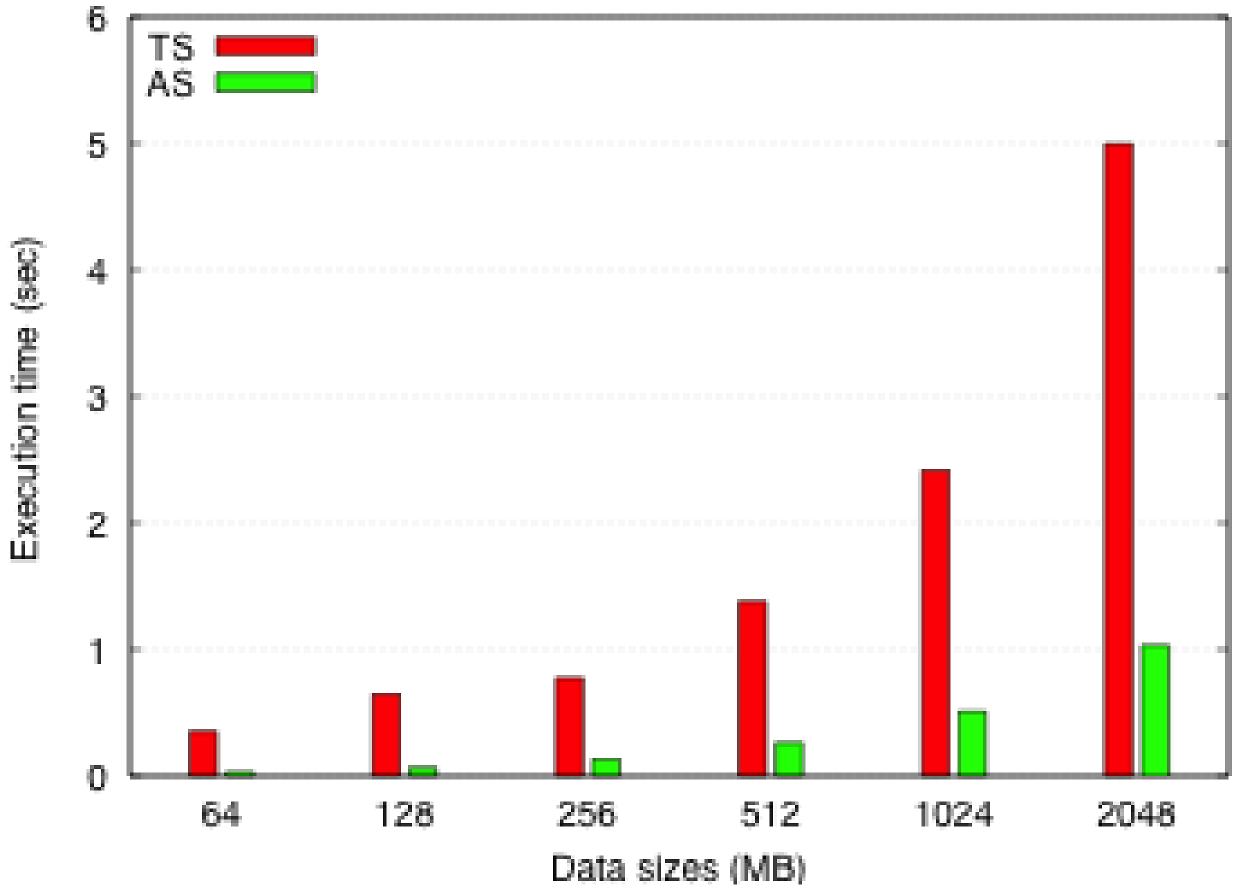
```
MPI_Info info;

MPI_Init();
MPI_Comm_rank();

MPI_Info_create (&info);
MPI_Info_set (info, "key", "val");

MPI_File_open ( …, info, … );
…

MPI_Info_free (&info);

MPI_Finalize();
```

<general MPI hint mechanism>

- Data type and operators are sufficient for simple operations, e.g., sum
- Some kernels might need more information to perform correct computation
  - Grep: string length per line (128), search pattern ("aaaaa")
  - K-means: number of dimension (10), number of clusters (20), threshold value (0.001), etc.

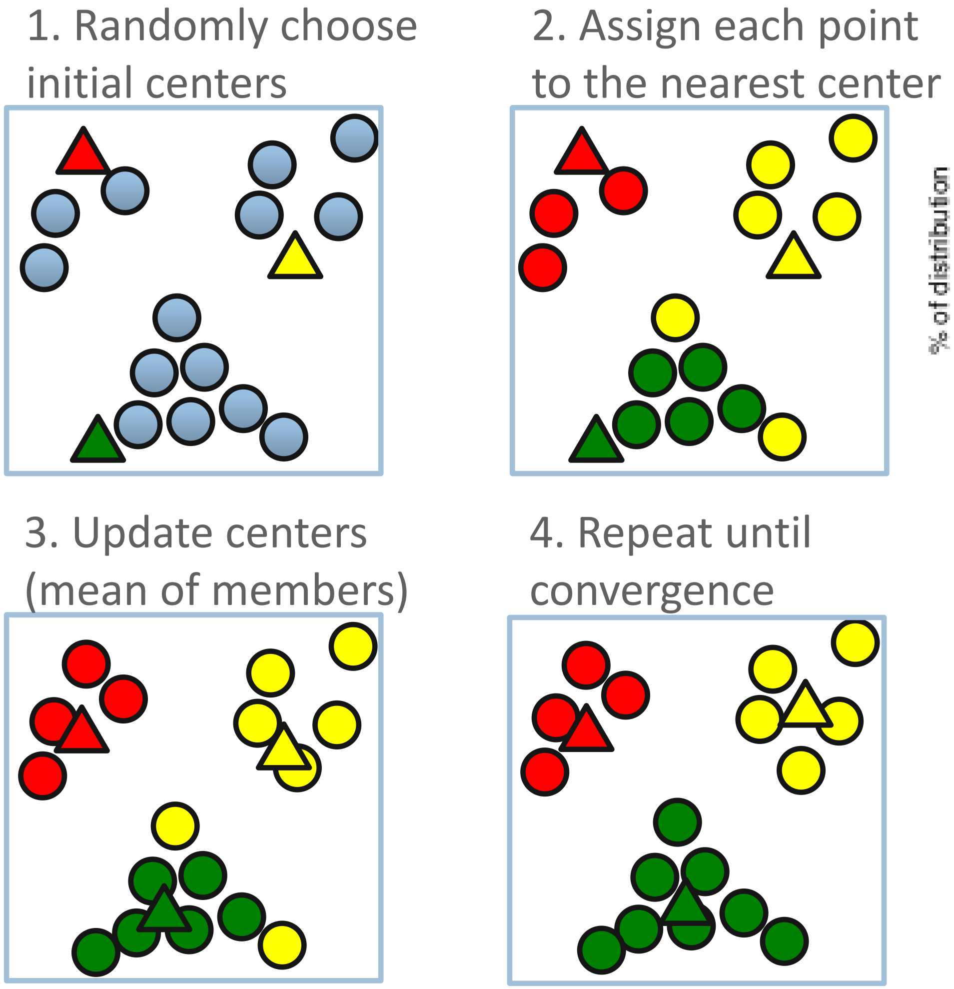
Our approach is scalable w.r.t the different data set sizes

SUM benchmark

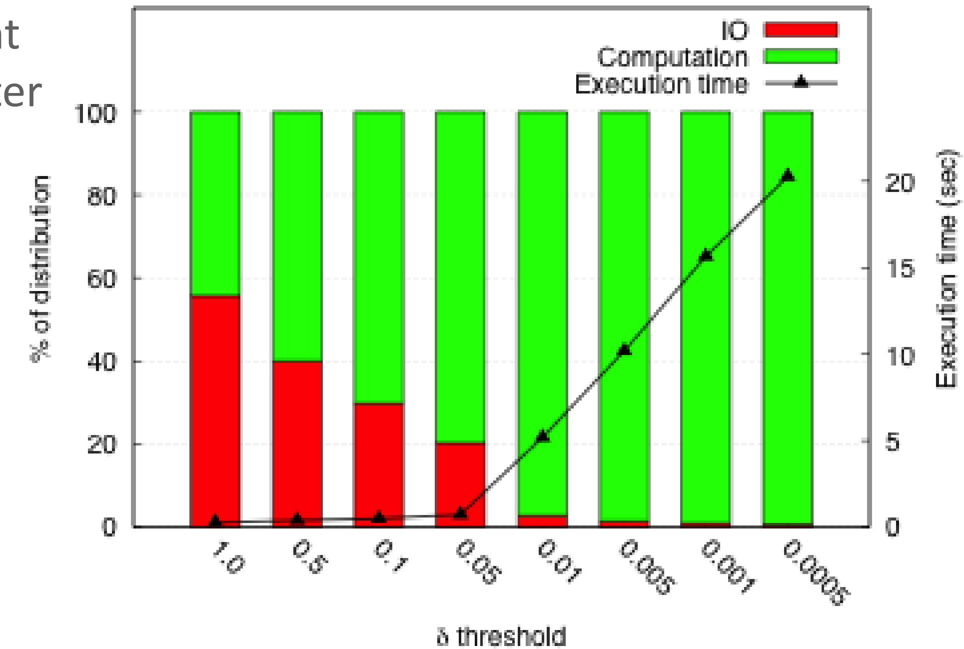


Fixed number of nodes: 4

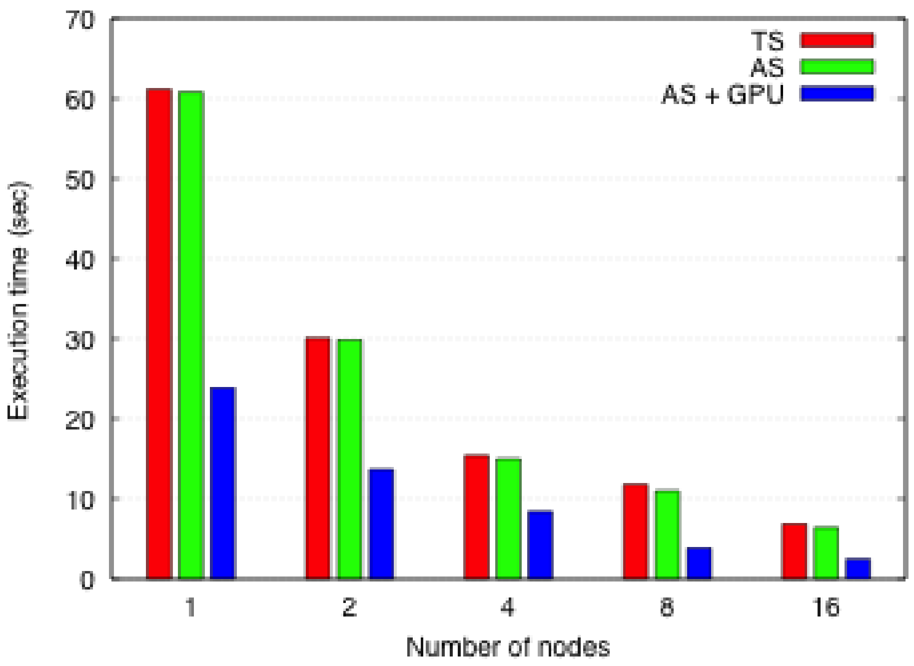
Data mining kernels can be compute intensive

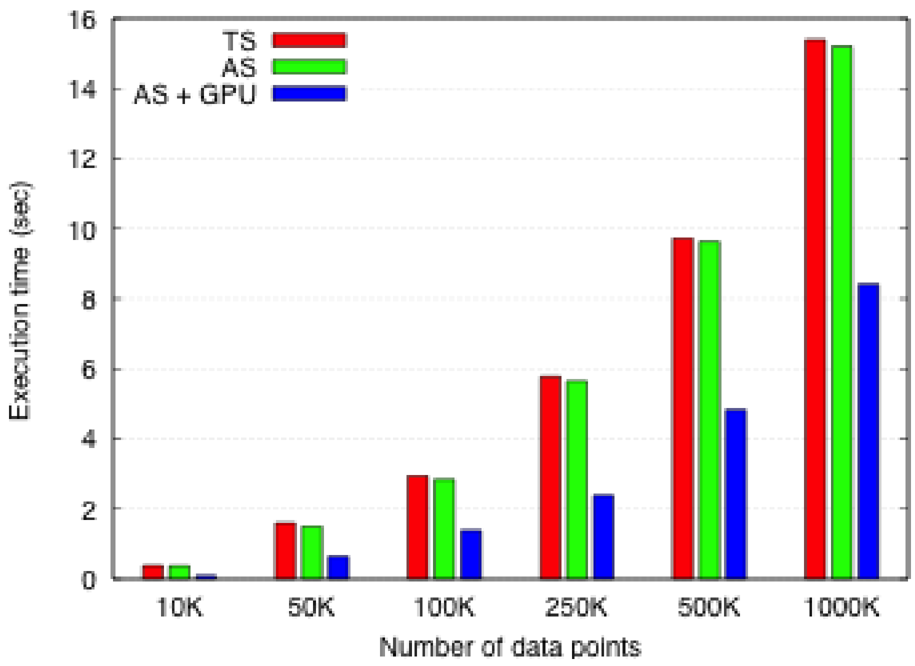


K-means clustering algorithm



Our approach is scalable w.r.t number of nodes to execute and data set size





Fixed data set size = 1M data points
Delta = 0.001
AS+GPU: active storage with GPU

Fixed # of nodes = 4
Delta = 0.001