Data Replication and Power Consumption in Data Grids

Karl Smith, Susan Vrbsky, Ming Lei, Jeff Byrd
University of Alabama
Tuscaloosa, AL

Introduction

- Grid computing – sharing of data and resources across multiple institutions
  - Large number of users, data, computation
- Data grid – managing and processing very large amount of data
  - BIRN – Biomedical Informatics Research Network
  - LHC – Large Hadron Collider
  - IVOA – International Virtual Observatory Alliance

Data Centers

- Data centers can meet the requirements of large-scale applications needing to process large amounts of data

Very energy inefficient

- Data centers inefficient – power and cooling
- Data centers consumed 1.5% of all electricity in U.S. in 2006, $4.5 B, 61B kWh
  - Predicted to double by 2011

Conserving Energy

- Ways to conserve energy
  - Power down CPU
  - Slow down CPU
  - Power down storage disks
    - Dynamic range is only 25% for disk drive
    - Disk devices have latency and energy penalty when transitioning from inactive to active mode
  - Store less data
- Will focus on storing less data
- Utilize Greener Grid

Data Aware Strategies

- Storing, managing, moving massive amounts of data also significant bottleneck
- Data aware strategies to reduce bottleneck
- Need data aware strategies to consider power efficiency - little effort thus far

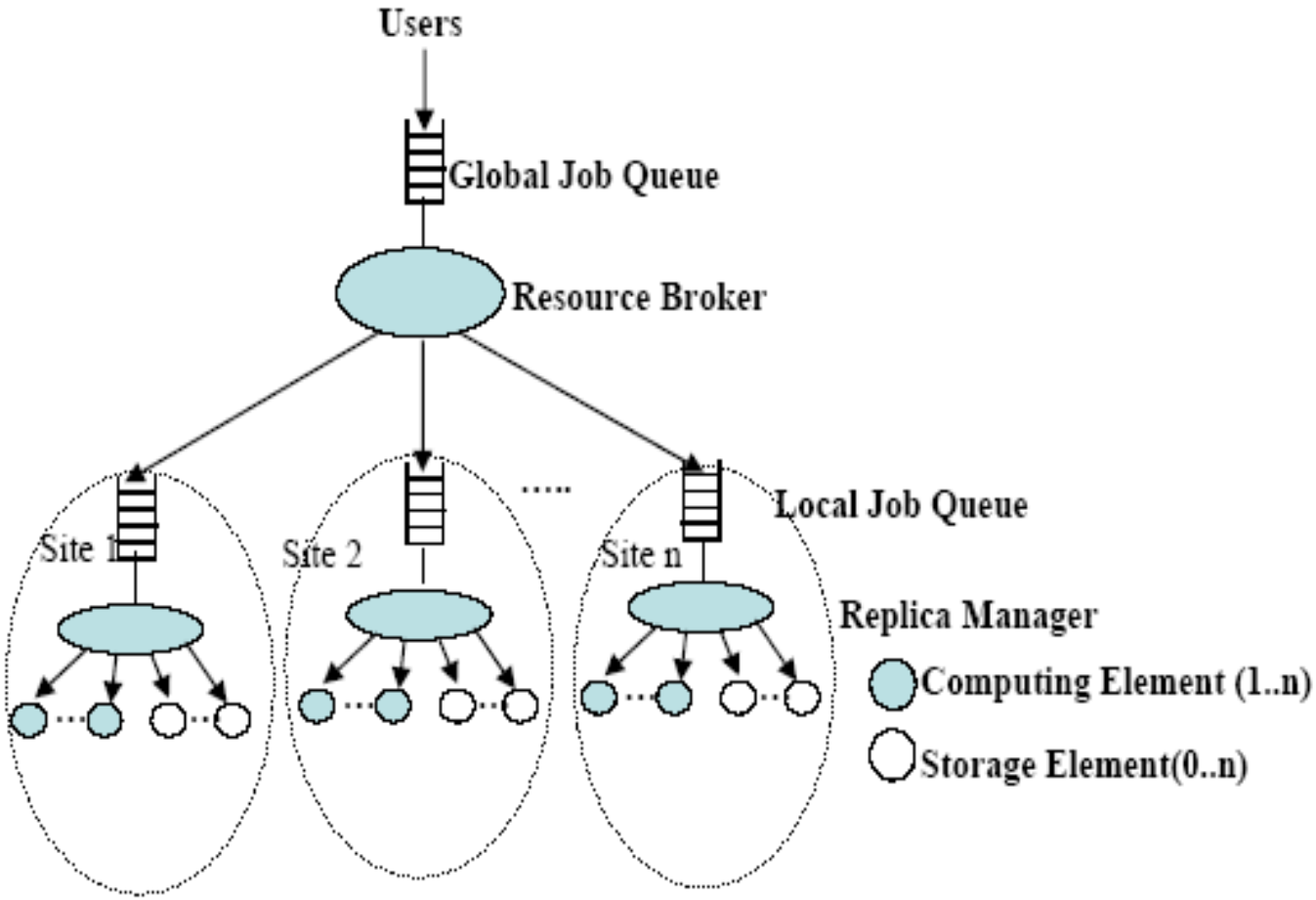
Our approach - Data Replication

- Replicate a file if needed in the future
- Due to dynamic nature of Grid user/system, difficult to make replica decisions to meet system availability goal
- Will focus on minimizing power consumed through smart data replication
- Will also utilize energy efficient cluster or grid

Related Work

- Economical model – replica decision based on auction protocol [Bell et al. 2003] e.g., replicate if used in future, unlimited storage
- Replicate fragments of files to increase data access performance and load balancing[Chang and Chen 2007]
- Analytical model to determine optimal number of replica servers, catalog servers, sizes to increase reliability [Schintke, Reinefeld 2003]
- Data aware CPU scheduling to improve performance and reliability [Kosar and Balman 2009]
- Green Grid metrics [2007] PUE and CDiE
- Energy proportional computing [Barroso and Holzle 2007]

Data Grid Architecture



CE – Computing Element
SE – Storage Element

Data Replication

- Can assume unlimited storage
  - Or
- If limited storage, use replacement strategies
    - LRU – least recently used
    - LFU – least frequently used
    - MRU – most recently used
  - Or
- Replicate based on new strategy

Sliding window replica protocol SWIN

- Propose – sliding window replica
  - Build sliding window – set of files to be used immediately in the future
  - Size bound by size of local SE
  - Includes all files current job will access and distinct files from next arriving jobs
  - Sliding window slides forward one more file each time system finishes processing a file
  - Sliding window dynamic

Sliding window replica protocol

- Sum of size of all files < size of SE
- No duplicate files in sliding window
- Each file contained in the sliding window will be utilized before any file not contained in the sliding window

Algorithm : BuildSlidingWindow(Job_Curr, Q_Input, Size_SE, Q_Block)

Input:

Job_Curr: The current active Job
Q: Current Job Queue
Size_SE: The current Storage Element's capacity
Q_Block: the Blocked Job Queue

Output:

Sliding_window: the set of files which will be accessed in the future.

```
{
    Sliding_window = null;
    sum_size_sliding_window =0;
    Q=SortByArrivalTime(Q  union with Q_Block )
     // sort  all the jobs in Q by the job arrival time
     for each file_temp  in Job_Curr's unprocessed file list
     {
         if (File_temp is not in Sliding_window)
        {
           if (sum_size_sliding_window<Size_SE)
             {
                Sliding_window.add(File_temp)
                sum_size_sliding_window   +=File_temp.size
             }
            else
             {      return Sliding_window              }
         }
        else  {     continue   }
                //if the file exists in the sliding window, continue to next file
    }  //end for

while (!Q.isEmpty())
    {
         temp_job = Q.nextJob()    //get the next job in the job queue
        for each File_temp  in temp_job's accessing file list
        {
            if (File_temp is not in Sliding_window)
            {
                if (sum_size_sliding_window<Size_SE)
                 {
                    Sliding_window.add(file_temp)
                    sum_size_sliding_window   +=File_temp.size
                 }
                 else
                 {      return Sliding_window    }
            }
             else {    continue    }
             //if the file exists in the sliding window, continue to next file
        }     //end for
    }
   return Sliding_window
}
```

Algorithm : ReplicateRemoteFile(RemoteFile, F_S)

Input:

RemoteFile : The file which is accessed remotely.

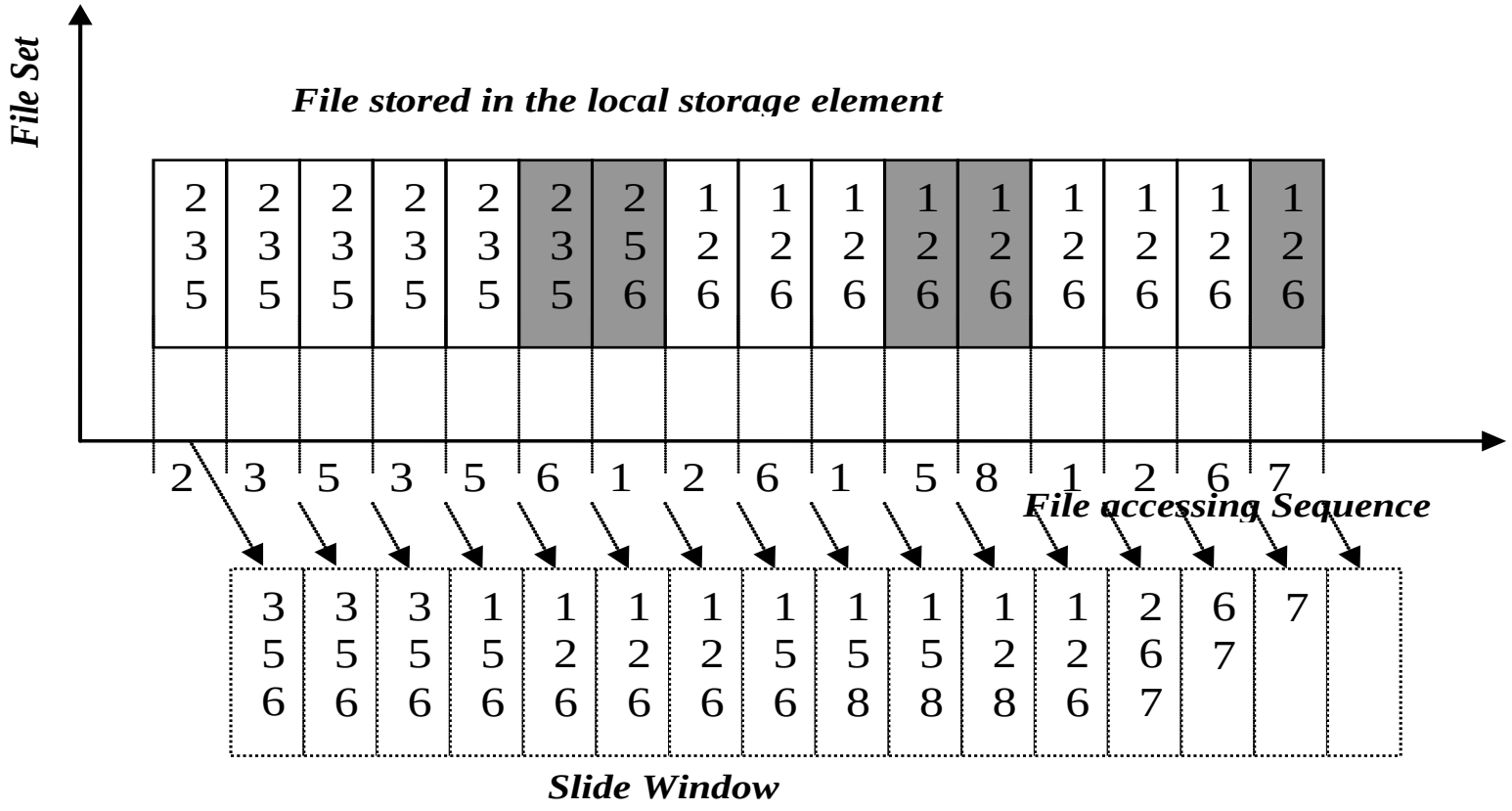
F_S : the current file set stored in the local storage element

Output

null

```
{
 SlidingWindow = BuildSlidingWindow()
//  call this function with correct parameters  to build the sliding window
if (RemoteFile does not exist in SlidingWindow)
 {      return     }
else
 {
        CandidateSize=0
        CandidateList=empty
        foreach tempFile in F_S
        {
                if (tempFile does not exist in SlidingWindow)
                 {
                      CandidateList.add(tempFile)
                       CandidateSize +=tempFile.size  }
                 if (CandidateSize > RemoteFile.size)
                   //there is enough space to replicate the remote file
                 {
                     RemoveFileInCandidateList() //remove all the candidate files
                      F_S.add(RemoteFile)  //Replicate the Remote File
                      return }
        }//end for
   }
}
```

Example

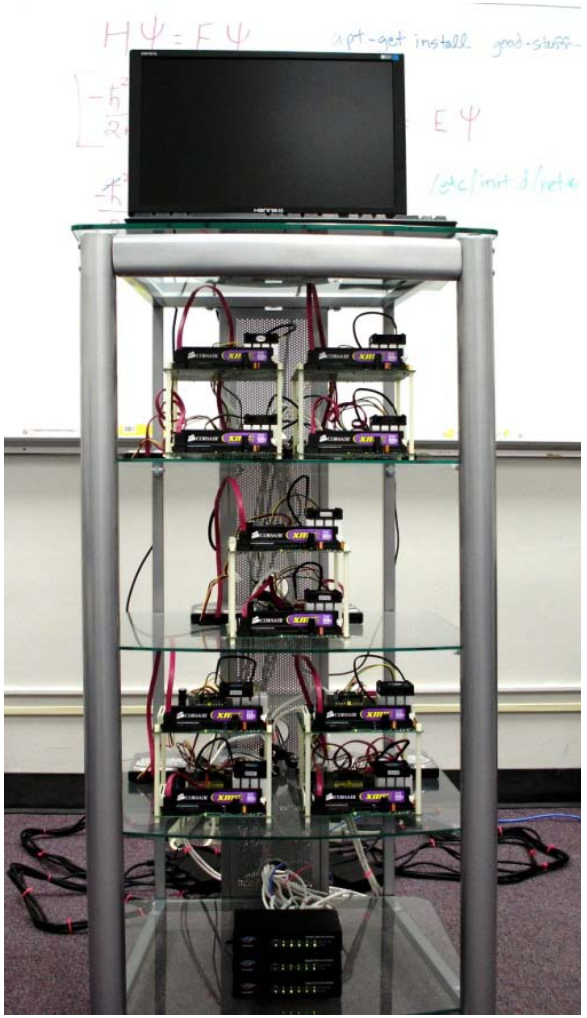

File Set
File stored in the local storage element
2 3 5
2 3 5
2 3 5
2 3 5
2 3 5
2 3 5
2 5 6
1 2 6
1 2 6
1 2 6
1 2 6
1 2 6
1 2 6
1 2 6
1 2 6
1 2 6
2 3 5 3 5 6 1 2 6 1 5 8 1 2 6 7
File accessing Sequence
3 5 6
3 5 6
3 5 6
1 5 6
1 2 6
1 2 6
1 2 6
1 5 6
1 5 8
1 5 8
1 2 8
1 2 6
2 6 7
6 7
7
Slide Window

SAGE Cluster Experiments

- To test performance of replica strategies
  - **Sage Cluster**
    - Built a green cluster (with lower energy mother boards)
    - Hardware purchased, put together
    - Can run off a single outlet
    - While booting, peak energy usage rates are approximately 430 Watts
    - While running, but idle, Celadon consumes 335 Watts with fan, 315 Watts without fan

SAGE Cluster

- Celadon Cluster – 10 nodes
  - Intel D201GLY2 mainboard with 1.2 GHz Celeron CPU
    - ($70 per)
  - 1 Gb 533 Mhz RAM
    - ($20 per)
  - 80 Gb SATA 3 hard drive
    - ($29 per)



SAGE Middleware

- Sage utilizes custom middleware developed at the University of Alabama, by myself and colleagues
- This middleware was designed to be modular allowing for easy implementation of replacement, replication, and scheduling algorithms as well as many other features

Description	Value
Number of files	*30*
File size	*40 MB*
Storage available at an SE	*10 files*
Number of files accessed by a job	*1-5*
Arrival distribution	*Poisson*
Request distribution	*Zipf*

Default experiment parameters

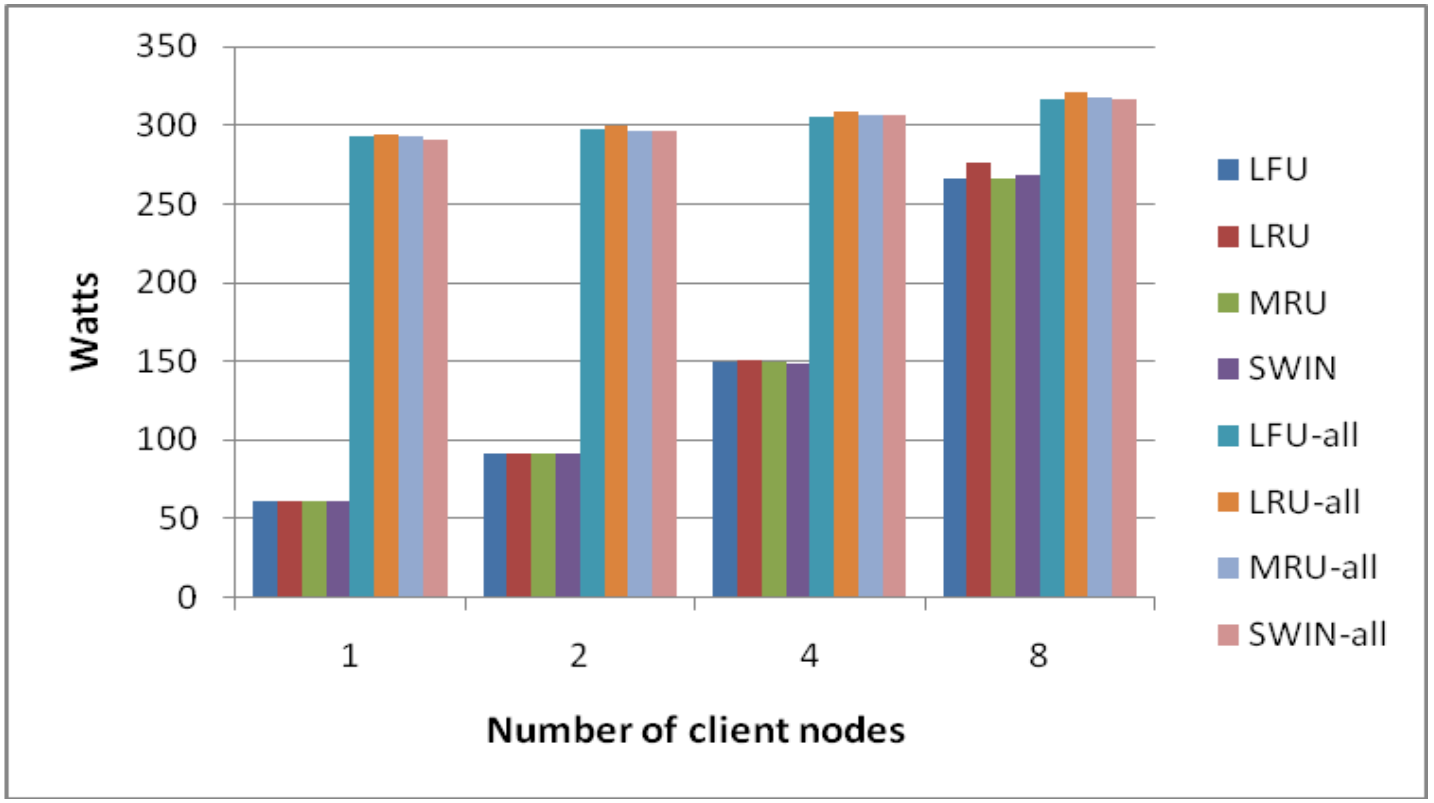
Experiments

- Measure:
  - Total Running Time
  - Average watts consumed as sampled periodically
    - Used power meter to measure
  - Average total watts consumed per job

Experiments

- Scheduling strategies
  - LRU, LFU, MRU, SWIN
- Number of client nodes
  - 1, 2, 4, 6, 8
- Number of nodes powered on
  - Necessary node vs. all nodes
- Size of files
  - 40 MB
- Assume all files initially available at server, but propagate during test

Power Consumed – continuous computation



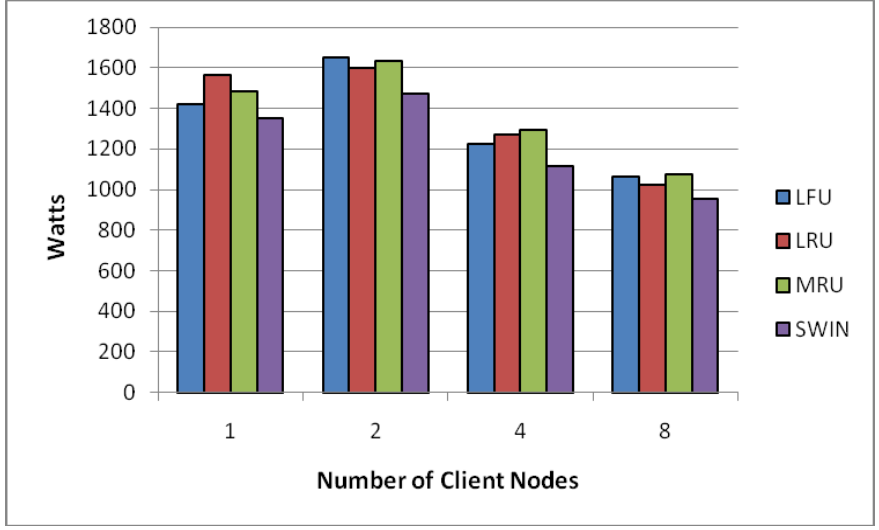
All nodes on vs. only nodes needed

Results

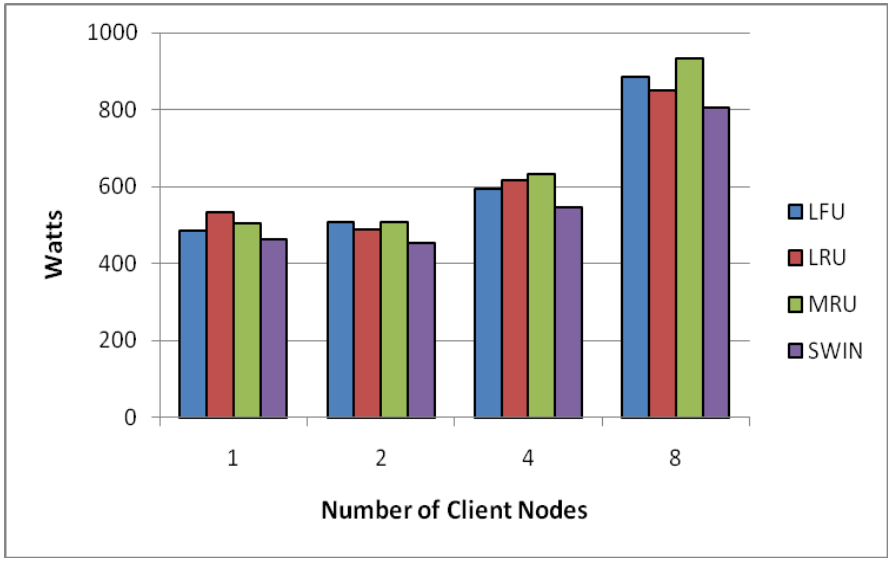
- SWIN uses less power than other strategies
- 2 clients, 79% more power on average when all clients on
- 6 clients, 51% more power on average when all clients on

Experiment - Number of client nodes

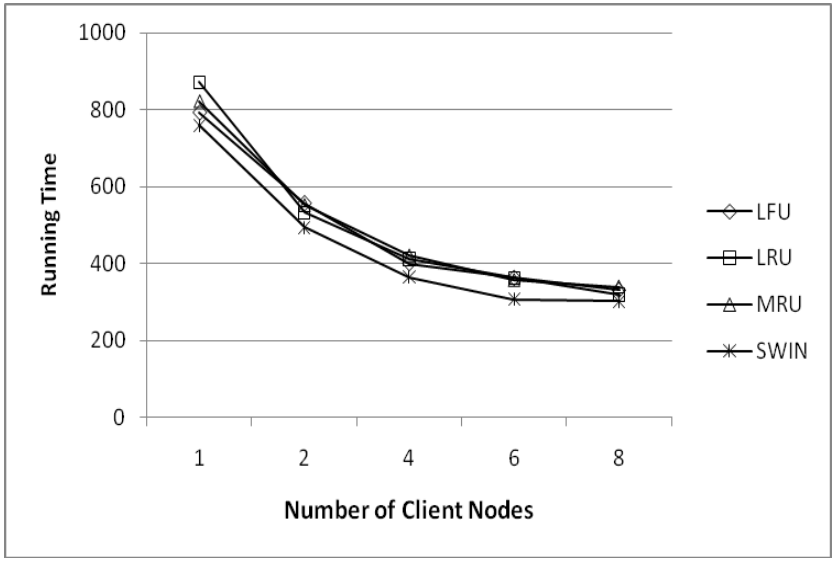
- 1,2,4,6,8 clients
- SE = 400MB
- 40 MB files, Zipf access, 100 jobs
- Average total watts per job
- All files initially available at server, distributed as replicated, can request copy from any node



Average Watts per job when all nodes powered



Average Watts per job when only required nodes powered on



Total Running Time

Results

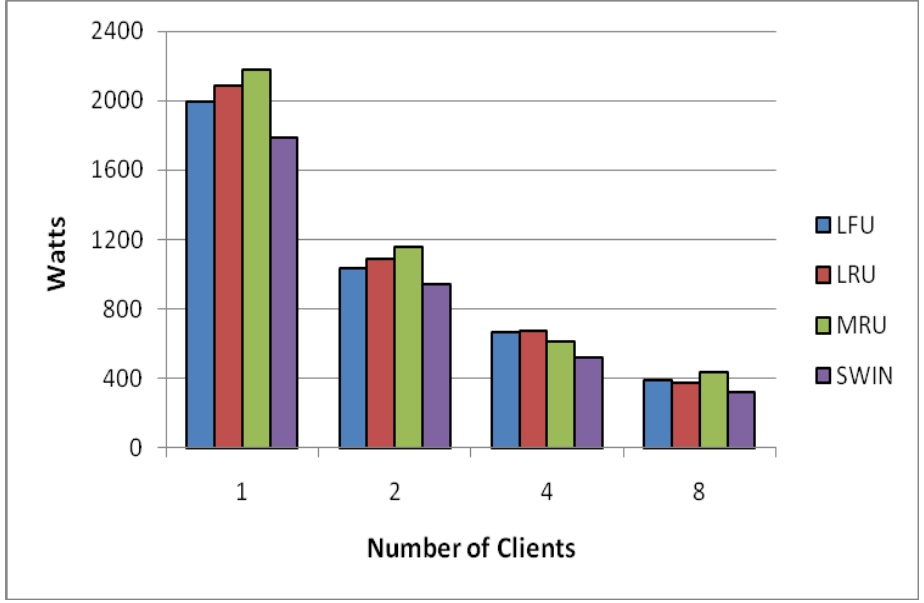
- All nodes powered on:
  - SWIN uses less power (12% less then competitors)
  - LRU requires most watts but due to shorter run time it does not require the most power
  - Jobs with 2 client nodes consumes most power
  - Power consumed decreases as number of clients increases
    - **Decrease in running time offsets increase in power**

Results

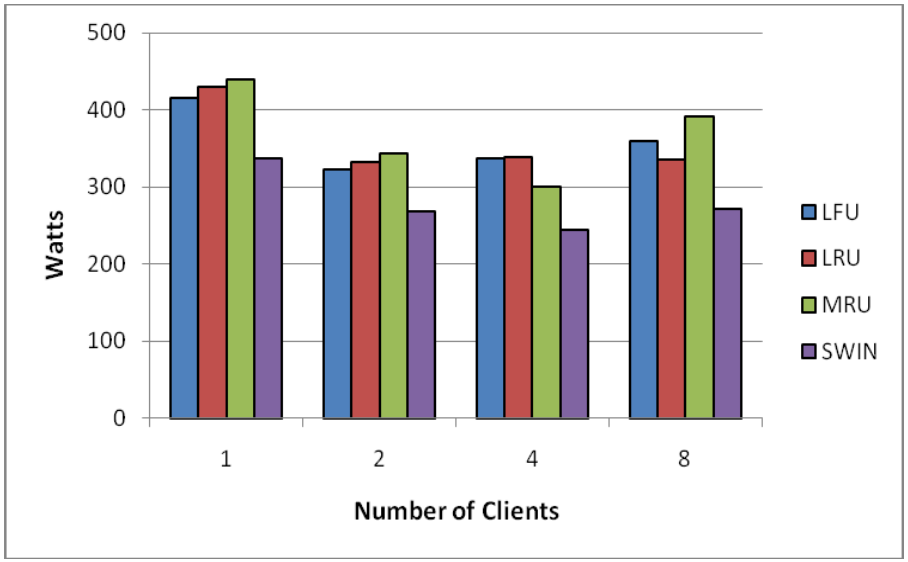
- Only required nodes powered on
  - Jobs requiring 1, 2 client nodes use least power, 8 client nodes most power
  - Increase in power consumed by increase in clients not offset by decrease in running time

Varied File Sizes

- Used 20 MB files
- Measure watts consumed
- All nodes vs. only required nodes powered on



Watts per job when all client nodes powered on



Watts per job when only required nodes powered on

Results

- Power consumed when all nodes powered on decreases dramatically as number of clients increases
- SWIN still uses least power (19-25% less)
- Power consumed when only needed nodes powered on
  - Most power consumed by 1 client and 8 client, less variation in power consumed
  - As number of client nodes increases, bottleneck increases and benefit from increasing number of clients lessened

Additional Results

- Decrease window size from 10 to 8
  - Running time and power consumed same as LRU
  - Less storage used but no increase in running time or power to process

Conclusions

- Proper file replacement can minimize running times and reduce power consumption by as much as 31%
  - Smaller storage can be used to lower power consumed
  - SWIN can reduce amount of storage by 25% and save on cost to cool center
- Amount of power consumed not always related to running time

Future Work

- Design additional storage strategies
- Data driven strategies to schedule jobs
- Powering down CPU in between processing jobs
- Implementation on larger scale clusters/grids utilizing virtualization on Fluffy (the University of Alabama's local cloud)

Questions?

Algorithm : BuildSlidingWindow(Job_Curr, Q_Input, Size_SE, Q_Block)

Input:

Job_Curr: The current active Job
Q: Current Job Queue
Size_SE: The current Storage Element's capacity
Q_Block: the Blocked Job Queue

Output:

Sliding_window: the set of files which will be accessed in the future.

```
{
    Sliding_window = null;
    sum_size_sliding_window =0;
    Q=SortByArrivalTime(Q  union with Q_Block )
     // sort  all the jobs in Q by the job arrival time
     for each file_temp  in Job_Curr's unprocessed file list
     {
         if (File_temp is not in Sliding_window)
        {
           if (sum_size_sliding_window<Size_SE)
             {
                Sliding_window.add(File_temp)
                sum_size_sliding_window   +=File_temp.size
             }
            else
             {      return Sliding_window              }
         }
        else  {     continue   }
                //if the file exists in the sliding window, continue to next file
    }  //end for

while (!Q.isEmpty())
    {
         temp_job = Q.nextJob()    //get the next job in the job queue
        for each File_temp  in temp_job's accessing file list
        {
            if (File_temp is not in Sliding_window)
            {
                if (sum_size_sliding_window<Size_SE)
                 {
                    Sliding_window.add(file_temp)
                    sum_size_sliding_window   +=File_temp.size
                 }
                 else
                 {      return Sliding_window    }
            }
             else {    continue    }
             //if the file exists in the sliding window, continue to next file
        }     //end for
    }
   return Sliding_window
}
```

Algorithm : ReplicateRemoteFile(RemoteFile, F_S)

Input:

RemoteFile : The file which is accessed remotely.

F_S : the current file set stored in the local storage element

Output

null

```
{
 SlidingWindow = BuildSlidingWindow()
//  call this function with correct parameters  to build the sliding window
if (RemoteFile does not exist in SlidingWindow)
 {      return     }
else
 {
        CandidateSize=0
        CandidateList=empty
        foreach tempFile in F_S
        {
                if (tempFile does not exist in SlidingWindow)
                 {
                      CandidateList.add(tempFile)
                       CandidateSize +=tempFile.size  }
                 if (CandidateSize > RemoteFile.size)
                   //there is enough space to replicate the remote file
                 {
                     RemoveFileInCandidateList() //remove all the candidate files
                      F_S.add(RemoteFile)  //Replicate the Remote File
                      return }
        }//end for
   }
}
```