MW : Master-Worker Middleware for Grids

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

NSF Shared Cyberinfrastructure (SCI) Division
Principal Investigators Meeting
February 19, 2004

MWCollaborators

- Jean-Pierre Goux
  - Northwestern University and Argonne National Lab
- Wen-Han Goh, Sanjeev Kulkarni, Miron Livny, Steve Wright, Mike Yoder
  - University of Wisconsin-Madison
- Jerry Shen
  - Lehigh University
- Bill Hart
  - Sandia National Lab

Outline

- MWMotivation (Why MW?)
  - MWHistory
  - Properties of Numerical Optimization Algorithms
- MWDesign (How MW?)
  - The MW API
  - The MW IPI (Infrastructure Programming Interface)
- MWSuccesses (What MW?)
  - The quadratic assignment problem—Solving nug30.
- MWFuture (When MW?)
  - Enhancements
  - New Applications

Outline

- MWMotivation (**Why** MW?)
  - MWHistory
  - Properties of Numerical Optimization Algorithms
- MWDesign (**How** MW?)
  - The MW API
  - The MW IPI (Infrastructure Programming Interface)
- MWSuccesses (**What** MW?)
  - The quadratic assignment problem—Solving nug30.
- MWFuture (**When** MW?)
  - Enhancements
  - New Applications

Outline

- MWMotivation (**Why** MW?)
  - MWHistory
  - Properties of Numerical Optimization Algorithms
- MWDesign (**How** MW?)
  - The MW API
  - The MW IPI (Infrastructure Programming Interface)
- MWSuccesses (**What** MW?)
  - The quadratic assignment problem—Solving nug30.
- MWFuture (**When** MW?)
  - Enhancements
  - New Applications

Outline

- MWMotivation (**Why** MW?)
  - MWHistory
  - Properties of Numerical Optimization Algorithms
- MWDesign (**How** MW?)
  - The MW API
  - The MW IPI (Infrastructure Programming Interface)
- MWSuccesses (**What** MW?)
  - The quadratic assignment problem—Solving nug30.
- MWFuture (**When** MW?)
  - Enhancements
  - New Applications

Outline

- MWMotivation (**Why** MW?)
  - MWHistory
  - Properties of Numerical Optimization Algorithms
- MWDesign (**How** MW?)
  - The MW API
  - The MW IPI (Infrastructure Programming Interface)
- MWSuccesses (**What** MW?)
  - The quadratic assignment problem—Solving nug30.
- MWFuture (**When** MW?)
  - Enhancements
  - New Applications

Outline

- MWMotivation (**Why** MW?)
  - MWHistory
  - Properties of Numerical Optimization Algorithms
- MWDesign (**How** MW?)
  - The MW API
  - The MW IPI (Infrastructure Programming Interface)
- MWSuccesses (**What** MW?)
  - The quadratic assignment problem—Solving nug30.
- MWFuture (**When** MW?)
  - Enhancements
  - New Applications

Outline

- MWMotivation (**Why** MW?)
  - MWHistory
  - Properties of Numerical Optimization Algorithms
- MWDesign (**How** MW?)
  - The MW API
  - The MW IPI (Infrastructure Programming Interface)
- MWSuccesses (**What** MW?)
  - The quadratic assignment problem—Solving nug30.
- MWFuture (**When** MW?)
  - Enhancements
  - New Applications

MWMotivation

- 1998, metaNEOS—Metacomputing Environments for Optimization.
  - NSF Grant to explore solving large scale numerical optimization problems on metacomputing (Grid computing) platforms
- Question:
  - Will existing (at that time) grid toolkits allow users to easily build grid-enabled optimization solvers?
- Answer:
  - To understand the tool requirements, we must understand the characteristics of optimization algorithms.

MWMotivation

- 1998, metaNEOS—Metacomputing Environments for Optimization.
    - NSF Grant to explore solving large scale numerical optimization problems on metacomputing (Grid computing) platforms
- Question:
    - Will existing (at that time) grid toolkits allow users to easily build grid-enabled optimization solvers?
- Answer:
    - To understand the tool requirements, we must understand the characteristics of optimization algorithms.

MWMotivation

- 1998, metaNEOS—Metacomputing Environments for Optimization.
  - NSF Grant to explore solving large scale numerical optimization problems on metacomputing (Grid computing) platforms
- Question:
  - Will existing (at that time) grid toolkits allow users to easily build grid-enabled optimization solvers?
- Answer:
  - To understand the tool requirements, we must understand the characteristics of optimization algorithms.

MWMotivation

- 1998, metaNEOS—Metacomputing Environments for Optimization.
  - NSF Grant to explore solving large scale numerical optimization problems on metacomputing (Grid computing) platforms
- Question:
  - Will existing (at that time) grid toolkits allow users to easily build grid-enabled optimization solvers?
- Answer:
  - To understand the tool requirements, we must understand the characteristics of optimization algorithms.

MWMotivation

- 1998, metaNEOS—Metacomputing Environments for Optimization.
  - NSF Grant to explore solving large scale numerical optimization problems on metacomputing (Grid computing) platforms
- Question:
  - Will existing (at that time) grid toolkits allow users to easily build grid-enabled optimization solvers?
- Answer:
  - To understand the tool requirements, we must understand the characteristics of optimization algorithms.

Broad Generalizations...

Optimization algorithms...

- Are iterative
  - Generally not "pleasantly parallel"
- Use data
  - Incrementally
  - "Optionally" (Potentially computed instead of shared)
- Are weakly synchronous
  - Can have their sychronization requirements reduced at a modest performance penalty
- Have a dynamic grain size
  - The computation can "easily" be broken into pieces of variable size.

Broad Generalizations...

Optimization algorithms...

- Are iterative
  - Generally not “pleasantly parallel”
- Use data
  - Incrementally
  - “Optionally” (Potentially computed instead of shared)
- Are weakly synchronous
  - Can have their sychronization requirements reduced at a modest performance penalty
- Have a dynamic grain size
  - The computation can “easily” be broken into pieces of variable size.

Broad Generalizations...

Optimization algorithms...

- Are iterative
  - Generally not "pleasantly parallel"
- Use data
  - Incrementally
  - "Optionally" (Potentially computed instead of shared)
- Are weakly synchronous
  - Can have their sychronization requirements reduced at a modest performance penalty
- Have a dynamic grain size
  - The computation can "easily" be broken into pieces of variable size.

Broad Generalizations...

Optimization algorithms...

- Are iterative
  - Generally not “pleasantly parallel”
- Use data
  - Incrementally
  - “Optionally” (Potentially computed instead of shared)
- Are weakly synchronous
  - Can have their sychronization requirements reduced at a modest performance penalty
- Have a dynamic grain size
  - The computation can “easily” be broken into pieces of variable size.

Broad Generalizations...

Optimization algorithms...

- Are iterative
  - Generally not “pleasantly parallel”
- Use data
  - Incrementally
  - “Optionally” (Potentially computed instead of shared)
- Are weakly synchronous
  - Can have their sychronization requirements reduced at a modest performance penalty
- Have a dynamic grain size
  - The computation can “easily” be broken into pieces of variable size.

Grid Toolkits. What did we want?

As numerical optimization researchers, we wanted a tool that would...

1. Be simple to use
2. Leverage a powerful platform
   - Like a Condor-provided computational grid
3. Be dynamic
   - Use resources as they became available
4. Be Fault-tolerant
   - Still compute the correct answer when machines fail
5. Be reusable for a large number of our algorithms!

Grid Toolkits. What did we want?

As numerical optimization researchers, we wanted a tool that would...

1. Be simple to use
2. Leverage a powerful platform
   - Like a Condor-provided computational grid.
3. Be dynamic
   - Use resources as they became available
4. Be Fault-tolerant
   - Still compute the correct answer when machines fail
5. Be reusable for a large number of our algorithms!

Grid Toolkits. What did we want?

As numerical optimization researchers, we wanted a tool that would...

1. Be simple to use
2. Leverage a powerful platform
   - Like a Condor-provided computational grid.
3. Be dynamic
   - Use resources as they became available
4. Be Fault-tolerant
   - Still compute the correct answer when machines fail
5. Be reusable for a large number of our algorithms!

Grid Toolkits. What did we want?

As numerical optimization researchers, we wanted a tool that would...

1. Be simple to use
2. Leverage a powerful platform
   - Like a Condor-provided computational grid.
3. Be dynamic
   - Use resources as they became available
4. Be Fault-tolerant
   - Still compute the correct answer when machines fail
5. Be reusable for a large number of our algorithms!

Grid Toolkits. What did we want?

As numerical optimization researchers, we wanted a tool that would...

1. Be simple to use
2. Leverage a powerful platform
   - Like a Condor-provided computational grid.
3. Be dynamic
   - Use resources as they became available
4. Be Fault-tolerant
   - Still compute the correct answer when machines fail
5. Be reusable for a large number of our algorithms!

Grid Toolkits. What did we want?

As numerical optimization researchers, we wanted a tool that would...

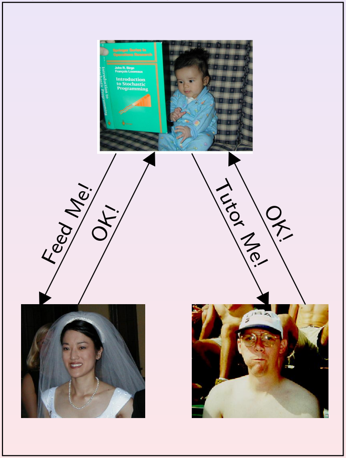
1. Be simple to use
2. Leverage a powerful platform
   - Like a Condor-provided computational grid.
3. Be dynamic
   - Use resources as they became available
4. Be Fault-tolerant
   - Still compute the correct answer when machines fail
5. Be reusable for a large number of our algorithms!

Master-Worker!

- Master assigns tasks to the workers
- Workers perform tasks, and report results back to master
- Workers do not communicate (except through the master)

- Simple!
- Dynamic/Fault-tolerant
- Reusable(!?)

Master-Worker

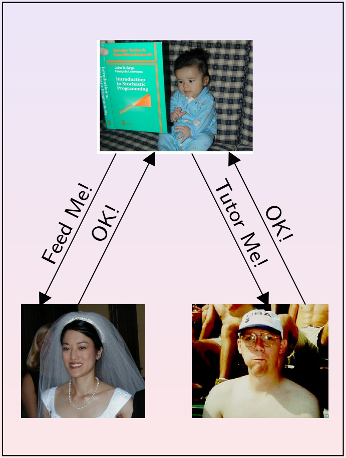


Master-Worker!

- Master assigns tasks to the workers
- Workers perform tasks, and report results back to master
- Workers do not communicate (except through the master)

- Simple!
- Dynamic/Fault-tolerant
- Reusable(!?)

Master-Worker

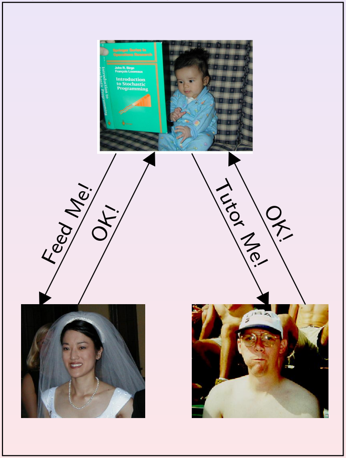


Master-Worker!

- Master assigns tasks to the workers
- Workers perform tasks, and report results back to master
- Workers do not communicate (except through the master)

- Simple!
- Dynamic/Fault-tolerant
- Reusable(!?)

Master-Worker

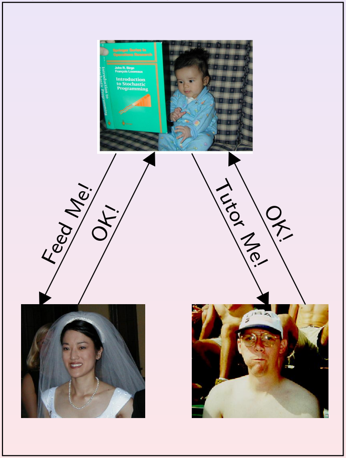


Master-Worker!

- Master assigns tasks to the workers
- Workers perform tasks, and report results back to master
- Workers do not communicate (except through the master)

- Simple!
- Dynamic/Fault-tolerant
- Reusable(!?)

Master-Worker



MW : A Master-Worker Grid Toolkit

- There are three abstraction in the master-worker paradigm: Master, Worker, and Task.
- MW is a software package that encapsulates these abstractions
  - API : C++ abstract classes
  - User writes 10 methods
  - The MWized code will transparently adapt to the dynamic and heterogeneous computing environment
- MW also has abstract layer to resource management and communications packages (an Infrastructure Programming Interface).
  - Condor/PVM
  - Condor/Files
  - Static/MPI
  - Single processor

MW : A Master-Worker Grid Toolkit

- There are three abstraction in the master-worker paradigm: Master, Worker, and Task.
- MW is a software package that encapsulates these abstractions
  - API : C++ abstract classes
  - User writes 10 methods
  - The MWized code will transparently adapt to the dynamic and heterogeneous computing environment
- MW also has abstract layer to resource management and communications packages (an Infrastructure Programming Interface).
  - Condor/PVM
  - Condor/Files
  - Static/MPI
  - Single processor

MW : A Master-Worker Grid Toolkit

- There are three abstraction in the master-worker paradigm: Master, Worker, and Task.
- MW is a software package that encapsulates these abstractions
  - API : C++ abstract classes
  - User writes 10 methods
  - The MWized code will transparently adapt to the dynamic and heterogeneous computing environment
- MW also has abstract layer to resource management and communications packages (an Infrastructure Programming Interface).
  - Condor/PVM
  - Condor/Files
  - Static/MPI
  - Single processor

MW API

- MWMaster
  - get_userinfo()
  - setup_initial_tasks()
  - pack_worker_init_data()
  - act_on_completed_task()
- MWTask
  - pack_work(), unpack_work()
  - pack_result(), unpack_result()
- MWWorker
  - unpack_worker_init_data()
  - execute_task()

MW API

- MWMaster
  - get_userinfo()
  - setup_initial_tasks()
  - pack_worker_init_data()
  - act_on_completed_task()
- MWTask
  - pack_work(), unpack_work()
  - pack_result(), unpack_result()
- MWWorker
  - unpack_worker_init_data()
  - execute_task()

MW API

- MWMaster
  - get_userinfo()
  - setup_initial_tasks()
  - pack_worker_init_data()
  - act_on_completed_task()
- MWTask
  - pack_work(), unpack_work()
  - pack_result(), unpack_result()
- MWWorker
  - unpack_worker_init_data()
  - execute_task()

But wait there's more!

- User-defined checkpointing of master
- (Rudimentary) Task Scheduling
  - MW assigns first task to first idle worker
  - Lists of tasks and workers can be arbitrarily ordered and reordered
  - User can set task rescheduling policies
- User-defined benchmarking
  - A (user defined) task is sent to each worker upon initialization
  - By accumulating normalized task CPU time, MW computes a performance statistic that is comparable between runs.

But wait there's more!

- User-defined checkpointing of master
- (Rudimentary) Task Scheduling
  - MW assigns first task to first idle worker
  - Lists of tasks and workers can be arbitrarily ordered and reordered
  - User can set task rescheduling policies
- User-defined benchmarking
  - A (user defined) task is sent to each worker upon initialization
  - By accumulating normalized task CPU time, MW computes a performance statistic that is comparable between runs.

But wait there's more!

- User-defined checkpointing of master
- (Rudimentary) Task Scheduling
  - MW assigns first task to first idle worker
  - Lists of tasks and workers can be arbitrarily ordered and reordered
  - User can set task rescheduling policies
- User-defined benchmarking
  - A (user defined) task is sent to each worker upon initialization
  - By accumulating normalized task CPU time, MW computes a performance statistic that is comparable between runs.

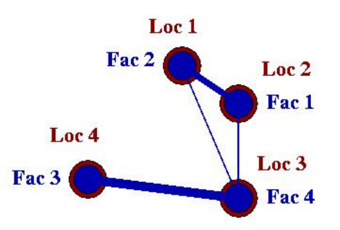
MW IPI

- Communication
  - `pack()`, `unpack()`, `send()`, `recv()`
  - Message buffer management routines
  - Changes in machine state are passed to master as tagged messages (HOSTADD, HOSTDELETE, etc.)
- Resource Management
  - `set_target_num_workers(int num_workers)`
  - `get_worker_info(MWWorkerID *)` : `MWWorkerID` class has members such as architecture, operating system, machine speed, etc.
  - `start_worker( MWWorkerID * )`

MW Applications

- MWFATCOP (Chen, Ferris, Linderoth) – A branch and cut code for linear integer programming
- MWMINLP (Goux, Leyffer, Nocedal) – A branch and bound code for nonlinear integer programming
- MWATR (Linderoth, Shapiro, Wright) – A trust-region-enhanced cutting plane code for linear stochastic programming and statistical verification of solution quality.
- MWQAP (Anstreicher, Brixius, Goux, Linderoth) – A branch and bound code for solving the quadratic assignment problem

The Quadratic Assignment Problem



- Assign facilities to locations
- QAP is NP-Hard
  - No known algorithm is "significantly better" than complete enumeration
  - Examining 10^9 configurations per second, for $n = 30$ would take 8,411,113,007,743,213 years, or $\approx 420,555$ Universe Lifetimes.

How Patient are You?

- If 8,411,113,007,743,213 years is a bit long to wait, you might try Branch and Bound.
  - Feasible solution $\Rightarrow$ upper bound
  - Relaxed problem $\Rightarrow$ lower bound

A detailed algorithmic description of branch and bound

1. Is solution to relaxed problem feasible?
   Yes? YAHOO!
   No? Break problem into smaller pieces. Goto 1.

- Conceptually, the there is a search tree than must be explored
- Different nodes are different *independent* searches
- Grid computing to the rescue!

How Patient are You?

- If 8,411,113,007,743,213 years is a bit long to wait, you might try Branch and Bound.
  - Feasible solution $\Rightarrow$ upper bound
  - Relaxed problem $\Rightarrow$ lower bound

A detailed algorithmic description of branch and bound

1. Is solution to relaxed problem feasible?
   Yes? YAHOO!
   No? Break problem into smaller pieces. Goto 1.

- Conceptually, the there is a search tree than must be explored
- Different nodes are different *independent* searches
- Grid computing to the rescue!

How Patient are You?

- If 8,411,113,007,743,213 years is a bit long to wait, you might try Branch and Bound.
  - Feasible solution ⇒ upper bound
  - Relaxed problem ⇒ lower bound

A detailed algorithmic description of branch and bound

1. Is solution to relaxed problem feasible?
   - Yes? YAHOO!
   - No? Break problem into smaller pieces. Goto 1.

- Conceptually, the there is a search tree than must be explored
- Different nodes are different *independent* searches
- Grid computing to the rescue!

How Patient are You?

- If 8,411,113,007,743,213 years is a bit long to wait, you might try Branch and Bound.
  - Feasible solution ⇒ upper bound
  - Relaxed problem ⇒ lower bound

A detailed algorithmic description of branch and bound

1. Is solution to relaxed problem feasible?
   Yes? YAHOO!
   No? Break problem into smaller pieces. Goto 1.

- Conceptually, the there is a search tree than must be explored
- Different nodes are different *independent* searches
- Grid computing to the rescue!

How Patient are You?

- If 8,411,113,007,743,213 years is a bit long to wait, you might try Branch and Bound.
  - Feasible solution ⇒ upper bound
  - Relaxed problem ⇒ lower bound

A detailed algorithmic description of branch and bound

1. Is solution to relaxed problem feasible?
   Yes? YAHOO!
   No? Break problem into smaller pieces. Goto 1.

- Conceptually, the there is a search tree than must be explored
- Different nodes are different *independent* searches
- Grid computing to the rescue!

How Patient are You?

- If 8,411,113,007,743,213 years is a bit long to wait, you might try Branch and Bound.
  - Feasible solution ⇒ upper bound
  - Relaxed problem ⇒ lower bound

A detailed algorithmic description of branch and bound

1. Is solution to relaxed problem feasible?

   Yes? YAHOO!

   No? Break problem into smaller pieces. Goto 1.

- Conceptually, the there is a search tree than must be explored
- Different nodes are different *independent* searches
- Grid computing to the rescue!

The Devil In The Details

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research
- We must avoid contention at the master
- Reduce arrival rate : Have machines work on a task for a sufficiently long time (Dynamic Grain Size)
- Increase service rate : Do *not* have workers pass back many nodes. Keep master's list of tasks small.
- Balancing efficiency considerations with search considerations was very important!
- We contend that with appropriate tuning, *many* algorithms can be shoehorned into the master-worker paradigm!
- MW can be a grid computing workhorse!

The Devil In The Details

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research
- We must avoid contention at the master
- Reduce arrival rate : Have machines work on a task for a sufficiently long time (Dynamic Grain Size)
- Increase service rate : Do *not* have workers pass back many nodes. Keep master's list of tasks small.
- Balancing efficiency considerations with search considerations was very important!
- We contend that with appropriate tuning, *many* algorithms can be shoehorned into the master-worker paradigm!
- MW can be a grid computing workhorse!

The Devil In The Details

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research
- We must avoid contention at the master
- Reduce arrival rate : Have machines work on a task for a sufficiently long time (Dynamic Grain Size)
- Increase service rate : Do *not* have workers pass back many nodes. Keep master's list of tasks small.
- Balancing efficiency considerations with search considerations was very important!
- We contend that with appropriate tuning, *many* algorithms can be shoehorned into the master-worker paradigm!
- MW can be a grid computing workhorse!

The Devil In The Details

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research
- We must avoid contention at the master
- Reduce arrival rate : Have machines work on a task for a sufficiently long time (Dynamic Grain Size)
- Increase service rate : Do *not* have workers pass back many nodes. Keep master's list of tasks small.
- Balancing efficiency considerations with search considerations was very important!
- We contend that with appropriate tuning, *many* algorithms can be shoehorned into the master-worker paradigm!
- MW can be a grid computing workhorse!

The Devil In The Details

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research
- We must avoid contention at the master
- Reduce arrival rate : Have machines work on a task for a sufficiently long time (Dynamic Grain Size)
- Increase service rate : Do *not* have workers pass back many nodes. Keep master's list of tasks small.
- Balancing efficiency considerations with search considerations was very important!
- We contend that with appropriate tuning, *many* algorithms can be shoehorned into the master-worker paradigm!
- MW can be a grid computing workhorse!

The Devil In The Details

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research
- We must avoid contention at the master
- Reduce arrival rate : Have machines work on a task for a sufficiently long time (Dynamic Grain Size)
- Increase service rate : Do *not* have workers pass back many nodes. Keep master's list of tasks small.
- Balancing efficiency considerations with search considerations was very important!
- We contend that with appropriate tuning, *many* algorithms can be shoehorned into the master-worker paradigm!
- MW can be a grid computing workhorse!

The Devil In The Details

- Fitting the B & B algorithm into the master-worker paradigm is not groundbreaking research
- We must avoid contention at the master
- Reduce arrival rate : Have machines work on a task for a sufficiently long time (Dynamic Grain Size)
- Increase service rate : Do *not* have workers pass back many nodes. Keep master's list of tasks small.
- Balancing efficiency considerations with search considerations was very important!
- We contend that with appropriate tuning, *many* algorithms can be shoehorned into the master-worker paradigm!
- MW can be a grid computing workhorse!

The Holy Grail

- nug30 (a QAP instance of size 30) had been the "holy grail" of computational QAP research for $>$ 30 years
- In 2000, we set out to solve this problem
- Using a mathematically sophisticated and well-engineered algorithm, we still estimated that we would require 11 CPU years to solve the problem.

The nug30 Computational Grid

Number	Type	Location	How
96	SGI/Irix	Argonne	Glide-in (Condor-G)
414	Intel/Linux	Argonne	Hobble-in
1024	SGI/Irix	NCSA	Glide-in (Condor-G)
16	Intel/Linux	NCSA	Flocked
45	SGI/Irix	NCSA	Flocked
246	Intel/Linux	Wisconsin	Flocked
146	Intel/Solaris	Wisconsin	Flocked
133	Sun/Solaris	Wisconsin	Flocked
190	Intel/Linux	Georgia Tech	Flocked
96	Intel/Solaris	Georgia Tech	Flocked
54	Intel/Linux	Italy (INFN)	Flocked
25	Intel/Linux	New Mexico	Flocked
12	Sun/Solaris	Northwestern	Flocked
5	Intel/Linux	Columbia U.	Flocked
10	Sun/Solaris	Columbia U.	Flocked

NUG30 is solved!

14, 5, 28, 24, 1, 3, 16, 15, 10, 9, 21, 2, 4, 29, 25, 22, 13, 26, 17, 30, 6, 20, 19, 8, 18, 7, 27, 12, 11, 23

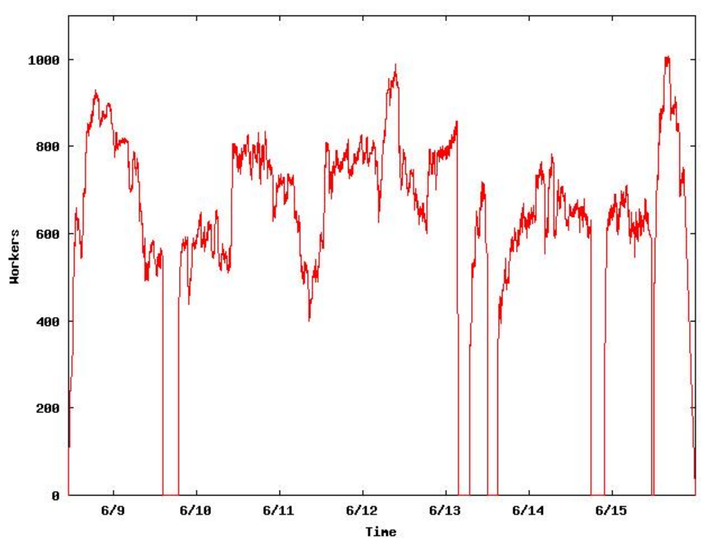
Wall Clock Time:	6:22:04:31
Avg. # Machines:	653
CPU Time:	≈ 11 years
Nodes:	11,892,208,412
LAPs:	574,254,156,532
Parallel Efficiency:	92%

NUG30 is solved!

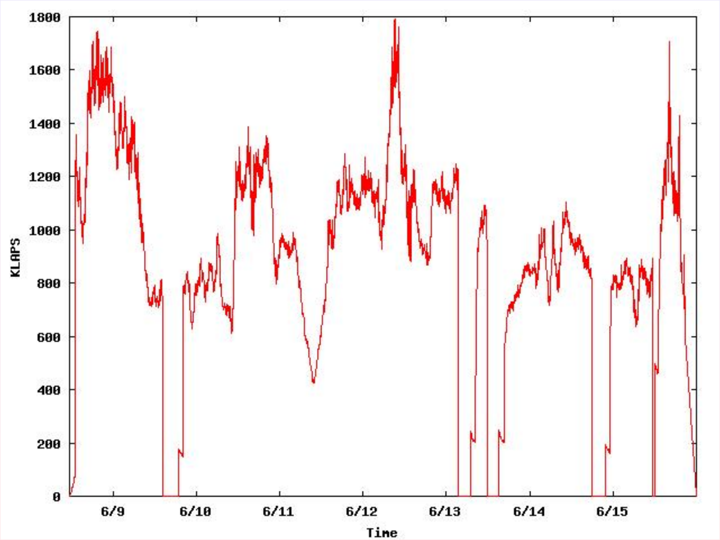
14, 5, 28, 24, 1, 3, 16, 15, 10, 9, 21, 2, 4, 29, 25, 22, 13, 26, 17, 30, 6, 20, 19, 8, 18, 7, 27, 12, 11, 23

Wall Clock Time:	6:22:04:31
Avg. # Machines:	653
CPU Time:	$\approx$ 11 years
Nodes:	11,892,208,412
LAPs:	574,254,156,532
Parallel Efficiency:	92%

Workers



KLAPS


KLAPS
1800
1600
1400
1200
1000
800
600
400
200
0
6/9
6/10
6/11
6/12
6/13
6/14
6/15
Time

MWRollout

- MW has been available from the Condor web page for some time.
  - Web: `http://www.cs.wisc.edu/condor/mw`
  - Mailing List: email `majordomo@cs.wisc.edu` with email body: `subscribe mw`
- A major focus of this proposal is to deploy MW as part of the NMI.
  - Improve robustness, documentation, and ease of use
  - Broaden and strengthen user base

MWRollout

- MW has been available from the Condor web page for some time.
  - Web: `http://www.cs.wisc.edu/condor/mw`
  - Mailing List: email `majordomo@cs.wisc.edu` with email body: `subscribe mw`
- A major focus of this proposal is to deploy MW as part of the NMI.
  - Improve robustness, documentation, and ease of use
  - Broaden and strengthen user base

New MWApplications

- Protein structure comparison (Sandia)
- Molecular Docking (UCSD/UW-Madison)
- Statistics: multicategory support vector machines (UW-Madison)
- Optimization: Multistage stochastic linear programming, nonconvex quadratic programming, mixed integer linear programming (Lehigh)
- Metaheuristics for combinatorial optimization (Polytech'Lille, France)
- Your application here!

New MWApplications

- Protein structure comparison (Sandia)
- Molecular Docking (UCSD/UW-Madison)
- Statistics: multicategory support vector machines (UW-Madison)
- Optimization: Multistage stochastic linear programming, nonconvex quadratic programming, mixed integer linear programming (Lehigh)
- Metaheuristics for combinatorial optimization (Polytech'Lille, France)
- Your application here!

MWEnhancements

- "Data Streaming"
  - Some master-worker type algorithms would benefit greatly from being able to keep a (low-bandwidth) data channel open between master and worker during execution of a task
  - Will be used in advanced distributed numerical optimization algorithms
- We want the enhancements to MW to be driven by its community of users!
  - Improved or dynamic load balancing?
  - Better interfaces (A Gui to steer/monitor?)
  - More RMComm implementations?
  - Other ideas???

MWEnhancements

- "Data Streaming"
  - Some master-worker type algorithms would benefit greatly from being able to keep a (low-bandwidth) data channel open between master and worker during execution of a task
  - Will be used in advanced distributed numerical optimization algorithms
- We want the enhancements to MW to be driven by its community of users!
  - Improved or dynamic load balancing?
  - Better interfaces (A Gui to steer/monitor?)
  - More RMComm implementations?
  - Other ideas???

Conclusions

- The master-worker paradigm can be effectively used to distribute many parallel scientific applications
  - Maybe yours too!
- The master-worker paradigm is nicely suited to a Grid implementation
  - We really believe that master-worker is the "right" paradigm for distributed computing on the Grid
- MW can make implementing master-worker algorithms for the Grid easier

Conclusions

- The master-worker paradigm can be effectively used to distribute many parallel scientific applications
  - Maybe yours too!
- The master-worker paradigm is nicely suited to a Grid implementation
  - We really believe that master-worker is the "right" paradigm for distributed computing on the Grid
- MW can make implementing master-worker algorithms for the Grid easier

Conclusions

- The master-worker paradigm can be effectively used to distribute many parallel scientific applications
  - Maybe yours too!
- The master-worker paradigm is nicely suited to a Grid implementation
  - We really believe that master-worker is the "right" paradigm for distributed computing on the Grid
- MW can make implementing master-worker algorithms for the Grid easier

Conclusions

- The master-worker paradigm can be effectively used to distribute many parallel scientific applications
  - Maybe yours too!
- The master-worker paradigm is nicely suited to a Grid implementation
  - We really believe that master-worker is the "right" paradigm for distributed computing on the Grid
- MW can make implementing master-worker algorithms for the Grid easier

The End!

We want YOU to join the MW community of users

http://www.cs.wisc.edu/condor/mw
http://www.mcs.anl.gov/metaneos/nug30