Implementing a Distributed Motion Planner

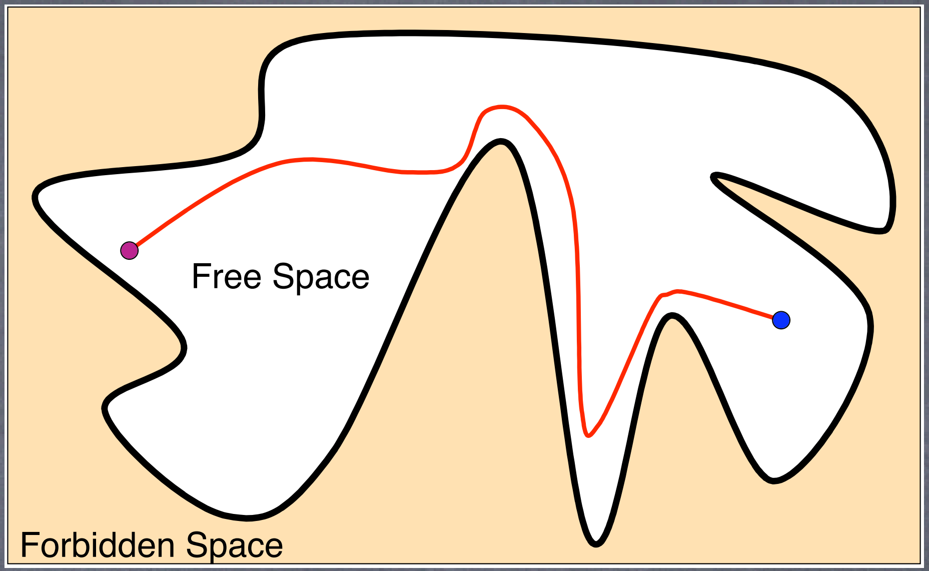
Presented by Jing Yang

Nov. 15, 2007

Agenda

- Review the distributed motion planner (DSRT)
- Implementation
  - Challenges
  - Asynchronous message passing
  - Classes
  - Experimental results
- Discussions

Motion Planning Basics

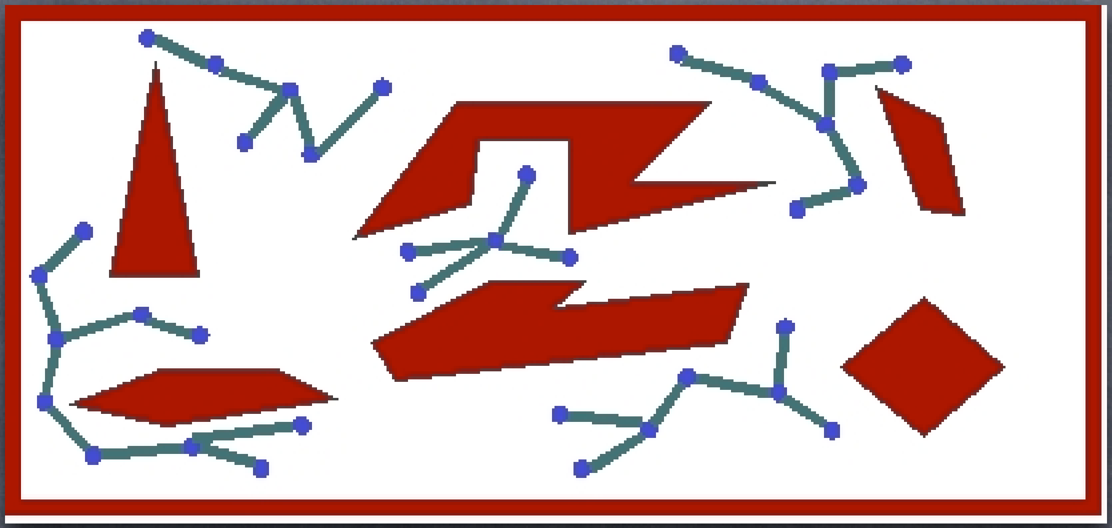


- Configuration space (C-Space) -- The space of all the configurations of the robot.
- Free C-Space -- The set of configurations at which the robot does not collide with any obstacles.
- Motion Planning -- Given two configurations of a robot, find a free path in the free C-Space that connects them.

Distributed SRT - Overview

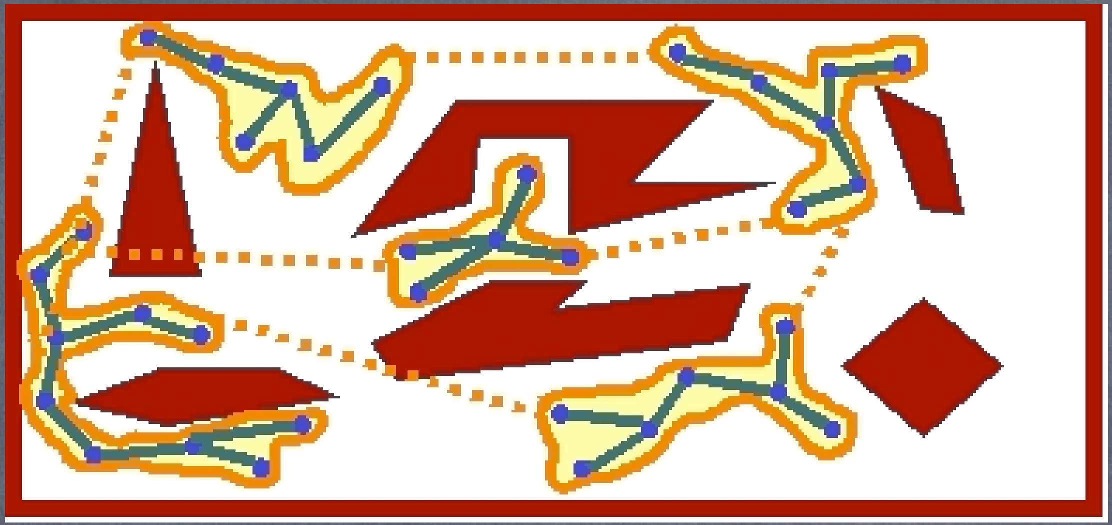
- A distributed algorithm using a master-client architecture
  - Clients {C1,...,Cc}: useful computations
  - Masters {M1,...,Mm}: schedule tasks

Algorithm (1)



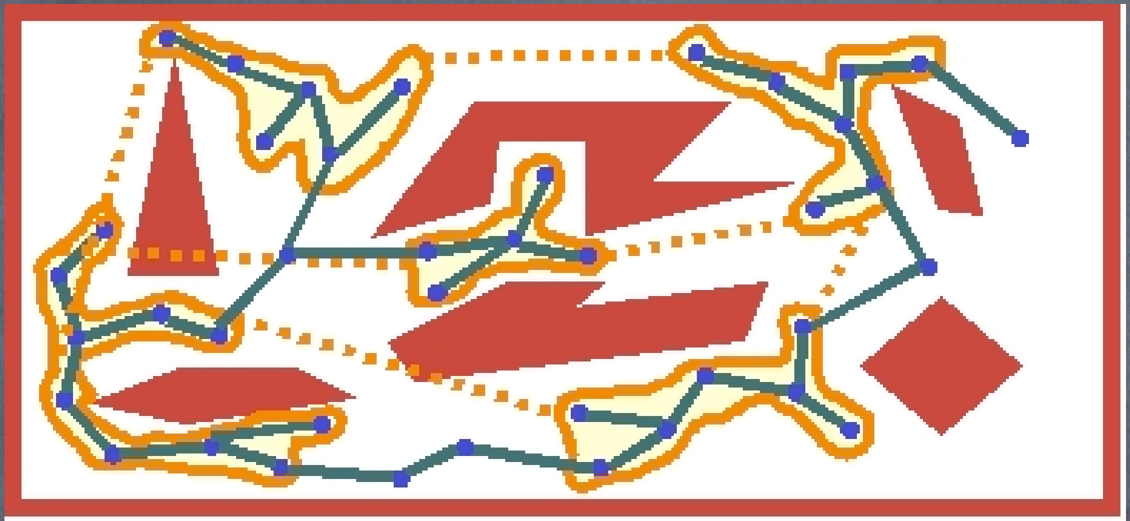
- Milestone Computations
- Candidate Edge Computations
- Edge Computations

Algorithm (2)



- Milestone Computations
- Candidate Edge Computations
- Edge Computations

Algorithm (3)

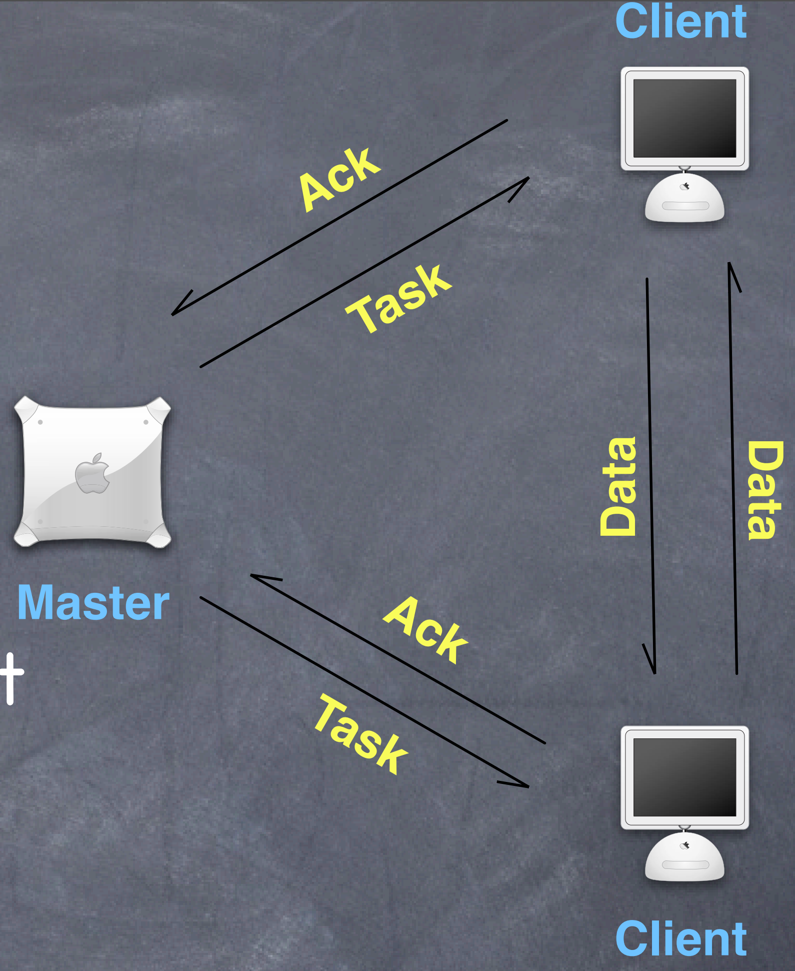


- Milestone Computations
- Candidate Edge Computations
- Edge Computations - concurrency issues!

Edge Computations

- Master assigns an edge for an available client
- Both milestones of the edge **must** be stored in the local memory of the chosen client
- Two cases:
    - Both milestones are currently owned by the client (simple)
    - One or neither is owned by the client (complex) - need other clients' help

Challenges for Implementing DSRT



- Complicated communications
  - Task assignment: master -> client
  - Ask for task: client -> master
  - Data sharing: client -> client
- Shared memory or message passing?
- Message passing: Synchronous or Asynchronous?
- Channel: one-to-one, one-to-many or many-to-one?

AsynchChannel

```
class AsyncChannel{
   private int numMessages;
   private Vector messages;
   private int receiverId;

   public synchronized void send(Object m){...}
   public synchronized Object receive(){...}}
```

- Message is queued if the receiver is busy
- Sender does not block
- Receiver blocks if there is no queued message

AsynchChannel: send & receive

```
public synchronized void send(Object m){
    if (m==null) throw new NullPointerException();
    numMessages++;
    messages.addElement(m);
    if (numMessages <= 0) notify(); //unblock the receiver}

public synchronized Object receive(){
    Object receivedMessage = null;
    numMessages--;
    if (numMessages < 0)
        try {wait();} //block the receiver
            catch (InterruptedException e) {}
    receivedMessage = messages.firstElement();
    messages.removeElementAt(0);
    return receivedMessage;}
```

Message types

Message class	Attributes	Flow
Available	Edge result; int senderId	Client -> Master

Edge	int src;
	int dst;
		Master -> Client

SendMilestoneTo	int toWhom;
	int milestoneId;
		Master -> Client

Milestone	int id;
	Object data;
		Client -> Client

Assumptions

- Channels shared by all the processors
- Error-free communication channels, i.e., no lost messages
- Messages can arrive in different order than they were sent
- Processors do not fail or halt

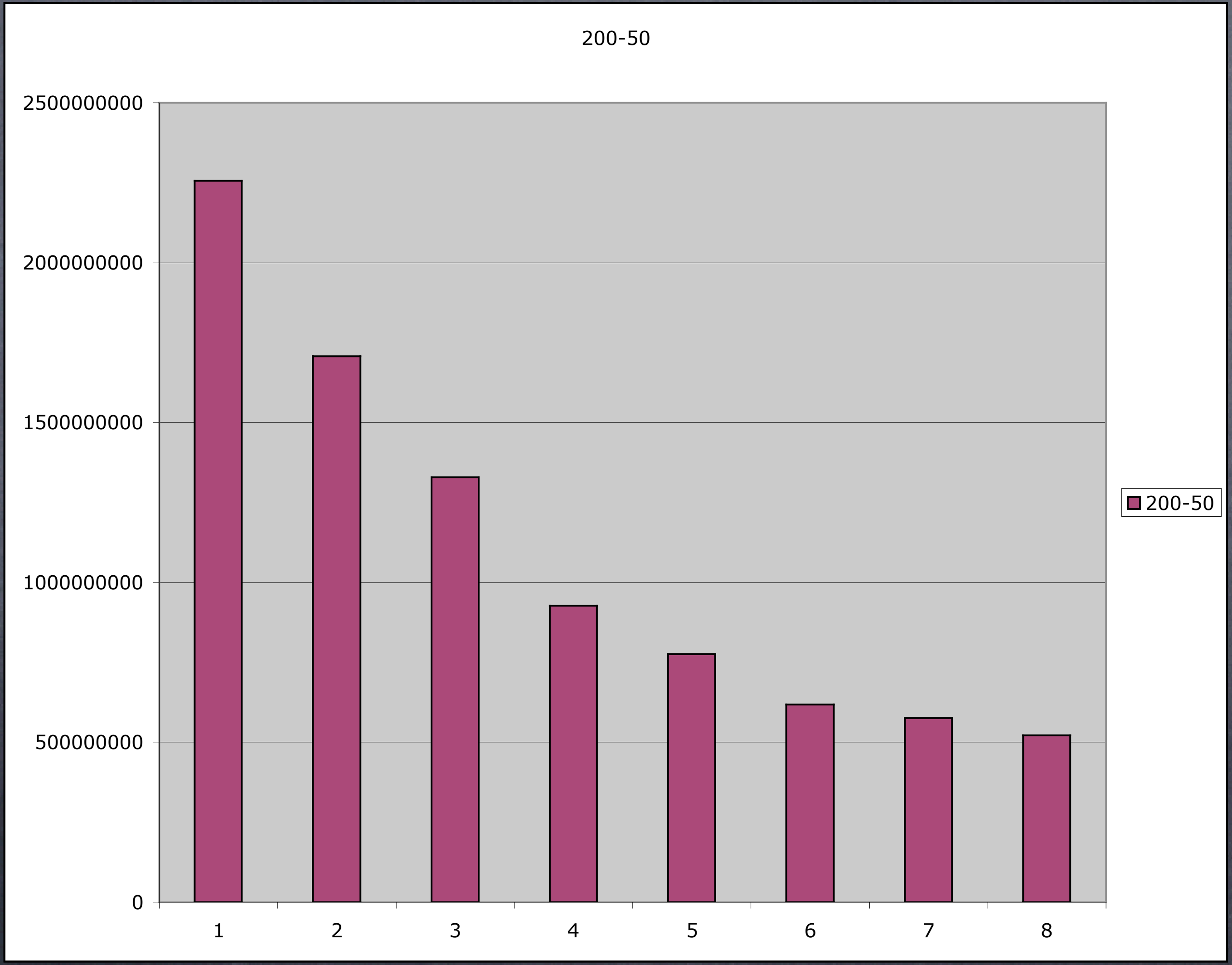
Master class

```
class Master extends Thread{
  int id; AsyncChannel[] channels; Edge[] edges; int numEdges;
  public void run(){
    while (numEdges>0){
      message=channels[id].receive();
      if (((Available)message).result != null) {numEdge--; update
      edges;}
      int cid=((Available)message).senderId;
      ...//select an edge e for cid
      channels[cid].send(e);
      ...//tell e.src and e.dst's owner x (if not cid) to send data to
      cid
      channels[x].send(new SendMilestoneTo(e.src, cid));
      ...
    }}}
```

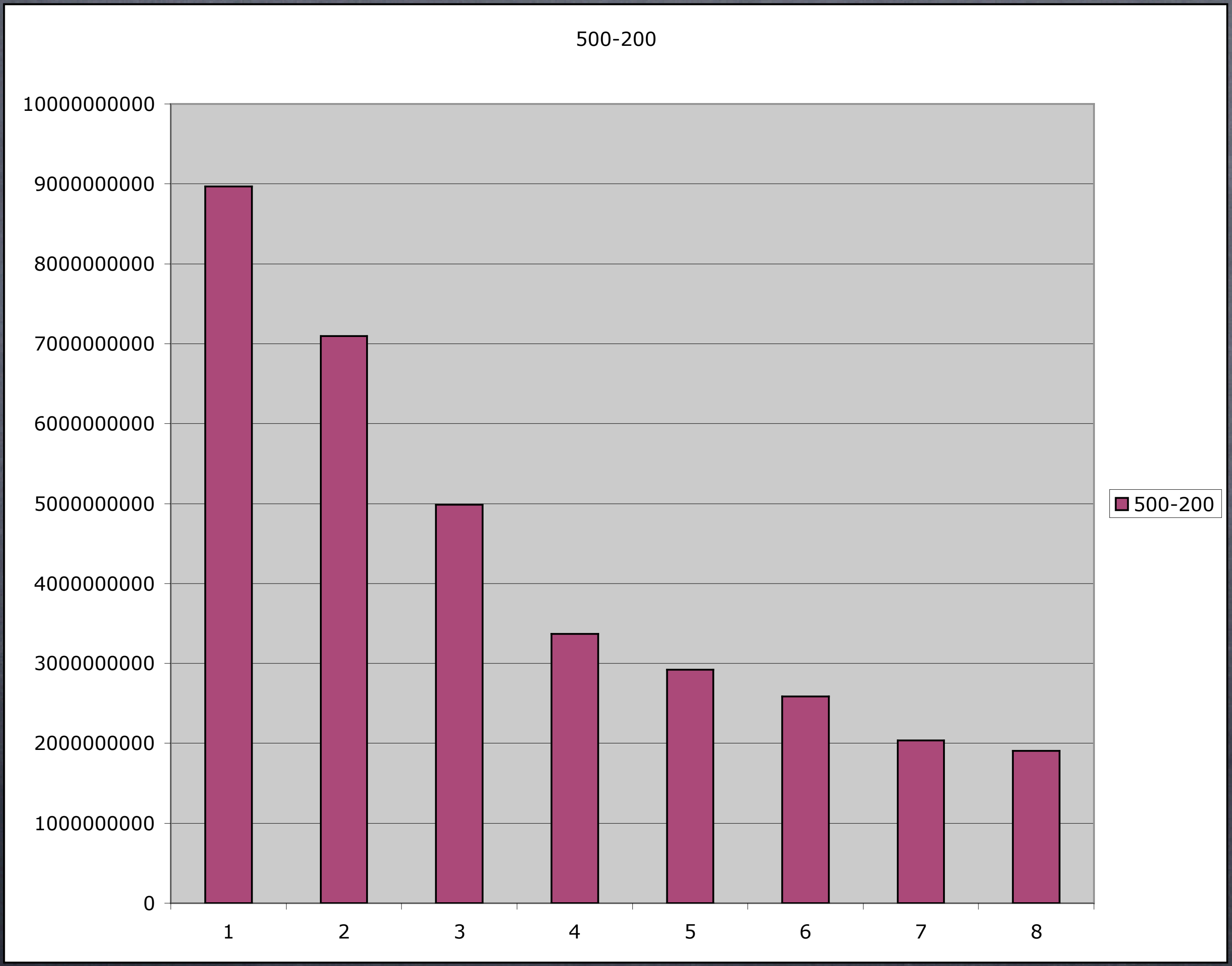
Client class

```
class Client extends Thread{
  int id; AsyncChannel[] channels;
  HashMap myMilestones; Edge currentJob;
  public void run(){
    while (true){
      message=channels[id].receive();
      if (message instanceof Edge) {currentJob=(Edge)message;}
      else if (message instanceof Milestone){
        myMilestones.put(message.id,message)}
      else if(message instanceof SendMilestoneTo){...//send milestone}
      ...//try connecting currentJob if both ends are in myMilestones
    }
  }
}
```

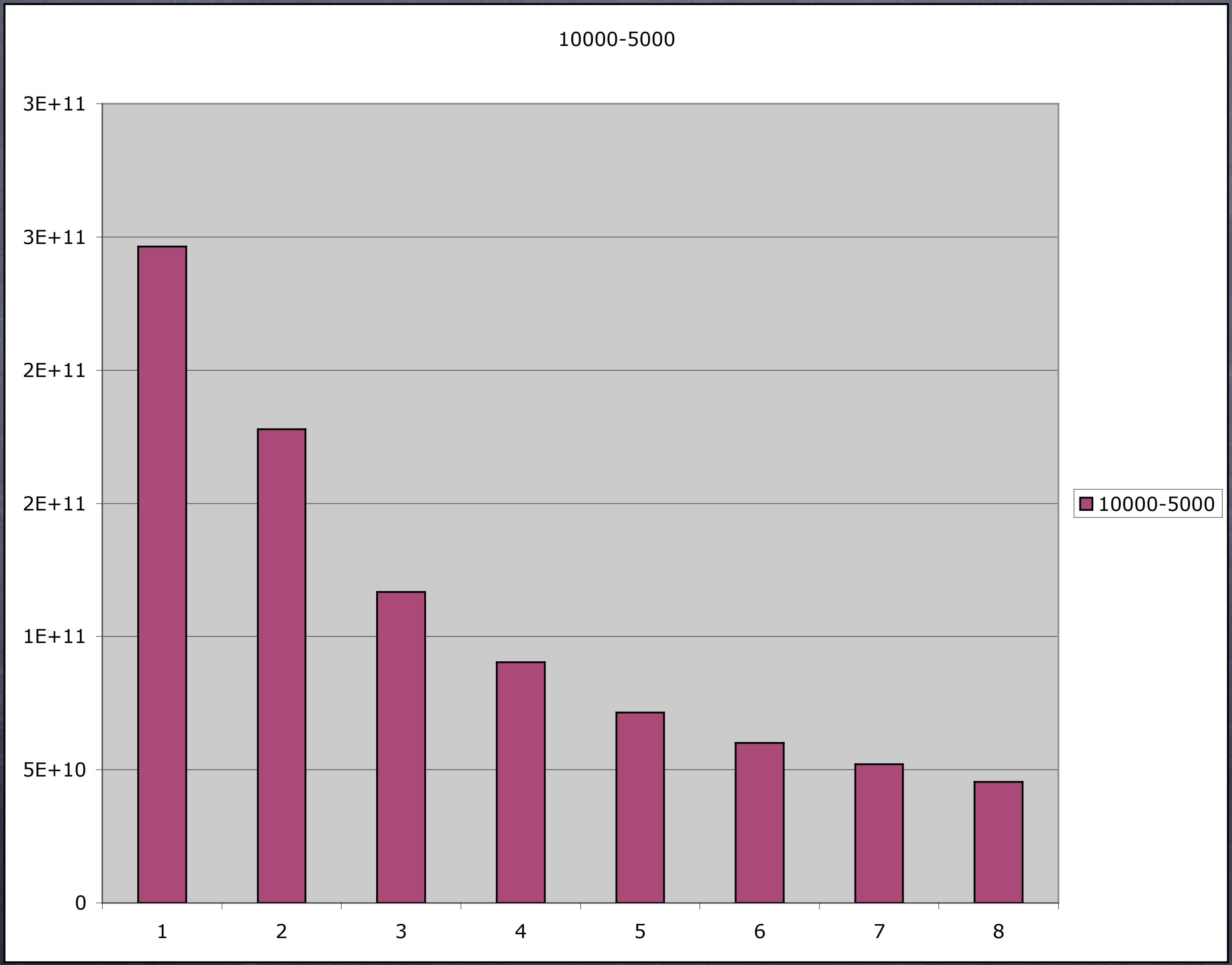
Experimental results (1)



Experimental results (2)



Experimental results (3)



Discussions

- Centralized design
- No "synchronized" method or object in the thread classes
- Possible optimizations:
    - Multiple masters
    - More than one job scheduled at a time
    - Cached memory (clients don't delete their temporary milestones immediately)

Questions?

Thank you!