MAPREDUCE OVER MOBILE DEVICES

PALAK SHAH

DIVYA RAMACHANDRAN

- **MapReduce System over Heterogeneous Mobile Devices**
  Authors: Peter R. Elespuru, Sagun Shakya, and Shivakant Mishra
  Publication: SEUS '09 Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems
  Link: http://dl.acm.org/citation.cfm?id=1694312

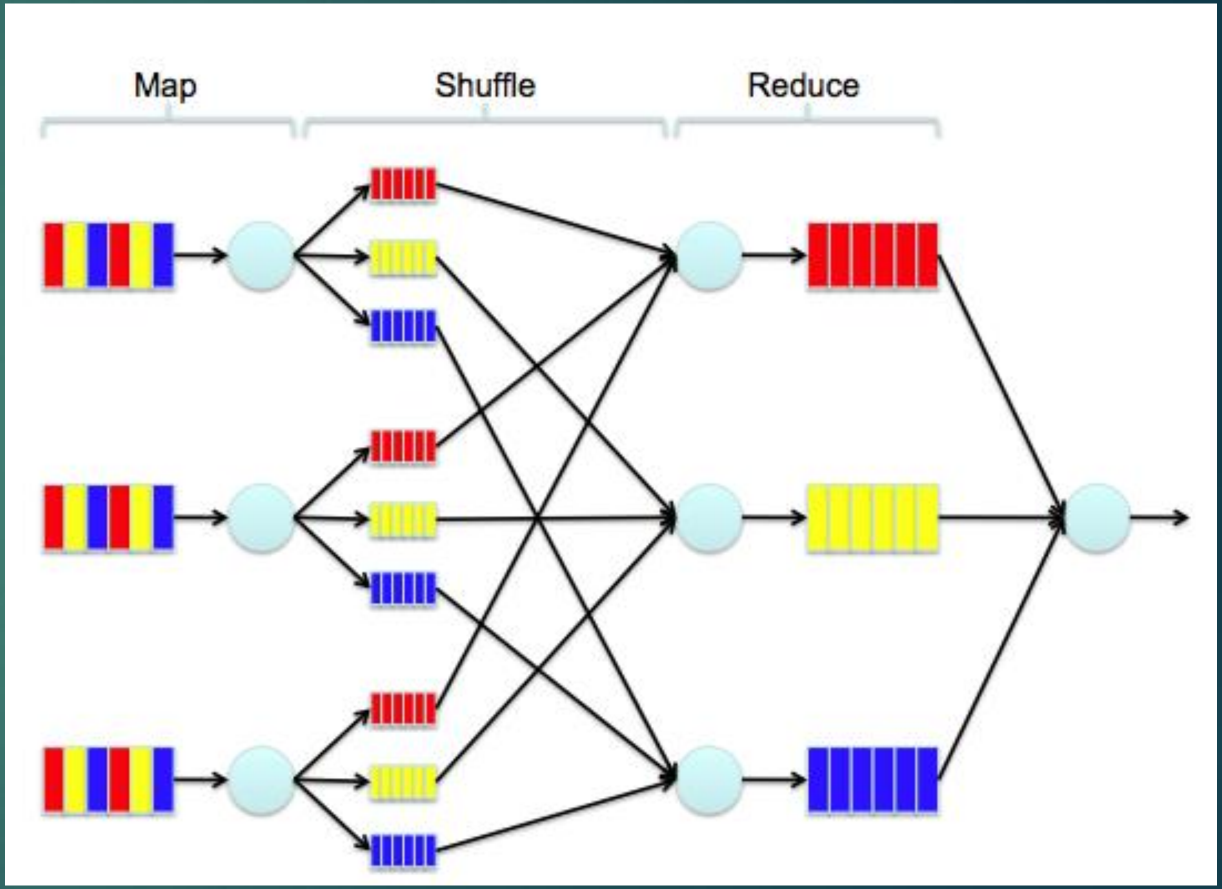
- **Scheduling for Real-Time Mobile MapReduce Systems**
  Authors: Adam J. Dou, Vana Kalogeraki, Dimitrios Gunopulos, Taneli Mielikäinen, Ville Tuulos
  Publication: DEBS '11 Proceedings of the 5th ACM international conference on Distributed event-based system
  Link: http://dl.acm.org/citation.cfm?id=2002305

Heterogeneous Mobile Device Map-Reduce System

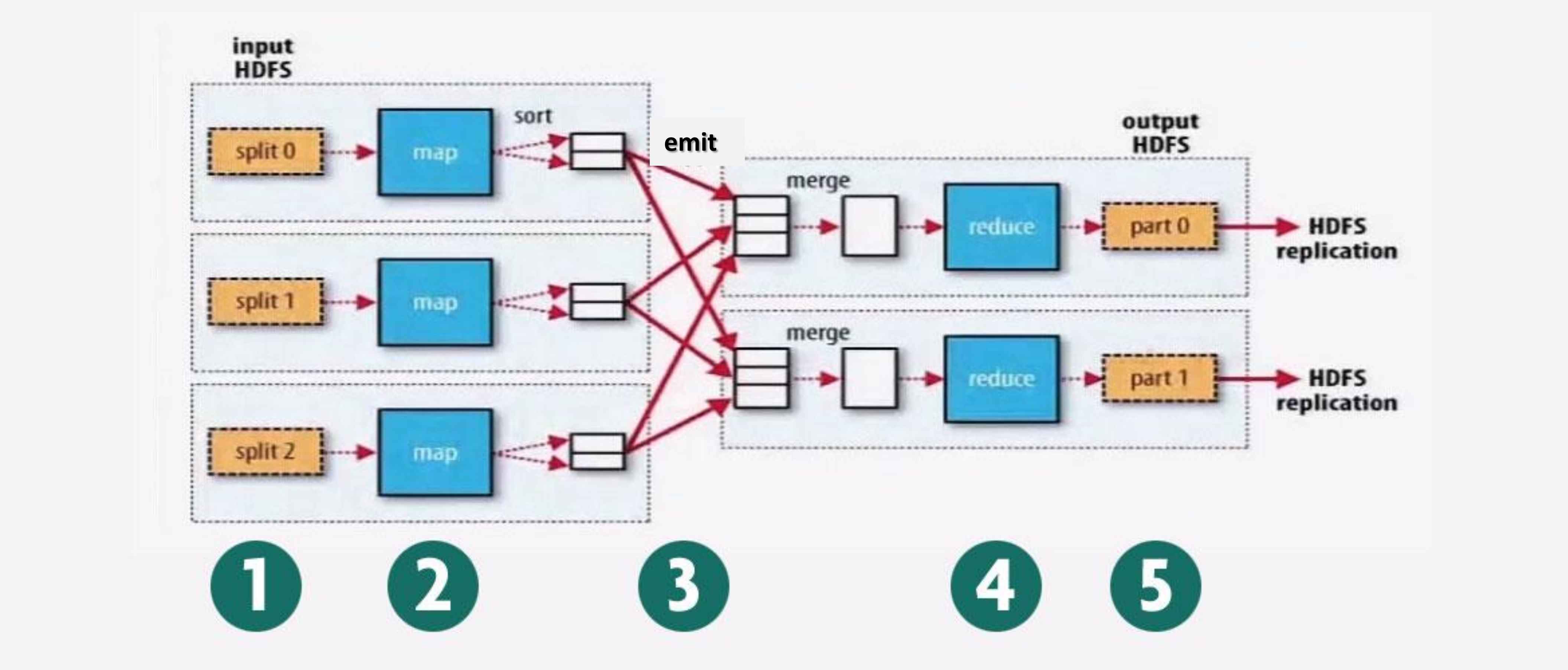
- Provide a mechanism for volunteers to participate in a smart phone distributed computational system
- Make use of this device pool to compute something and provide aggregate results
- Provide interesting results to interested parties and summarize them in a timely fashion considering the reliability of mobile devices and network communications

MapReduce

- Created at Google in 2004 by Jeffrey Dean and Sanjay Ghemawat
- Distributed Processing Algorithm
- Reduces large problem sets into small pieces
- Distributed tasks completed by cluster of devices
- Solves basically problems that are huge, but not hard
- Example – Indexing of documents for search



Map Reduce Example



http://blog.pivotal.io/pivotal/products/hadoop-101-programming-mapreduce-with-native-libraries-hive-pig-and-cascading

Related Projects

Some projects that allow interested users to surrender a portion of their desktop or laptop to a much larger computational goal:

- **SETI@Home**
  - Analyze data in search of extra terrestrial signals
- **Folding@Home**
  - Understand protein folding and related diseases

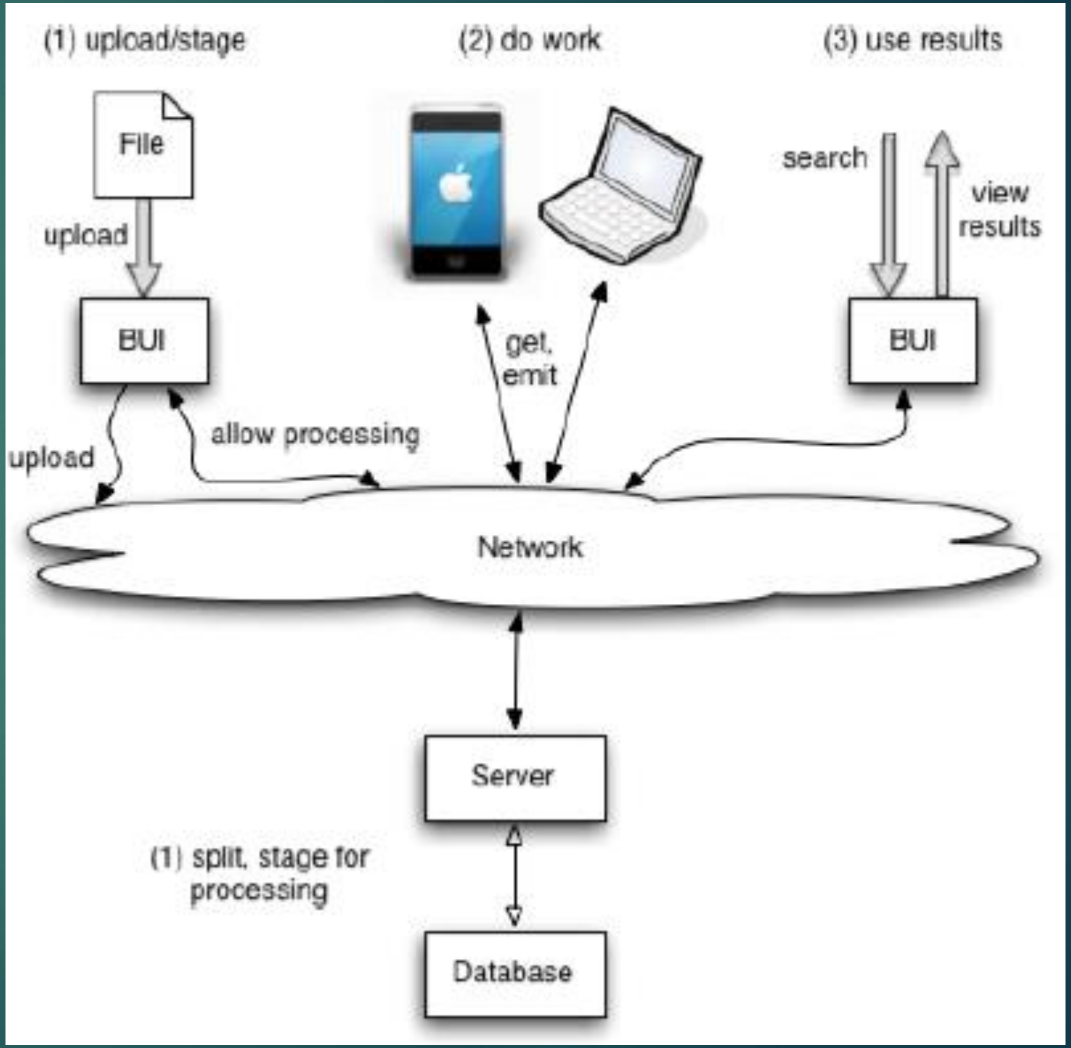
Limitations on Mobile Devices

- Only smart phones are computationally powerful enough for these applications
- Power usage
- Security Concerns
- Interference with traditional usage model as a phone

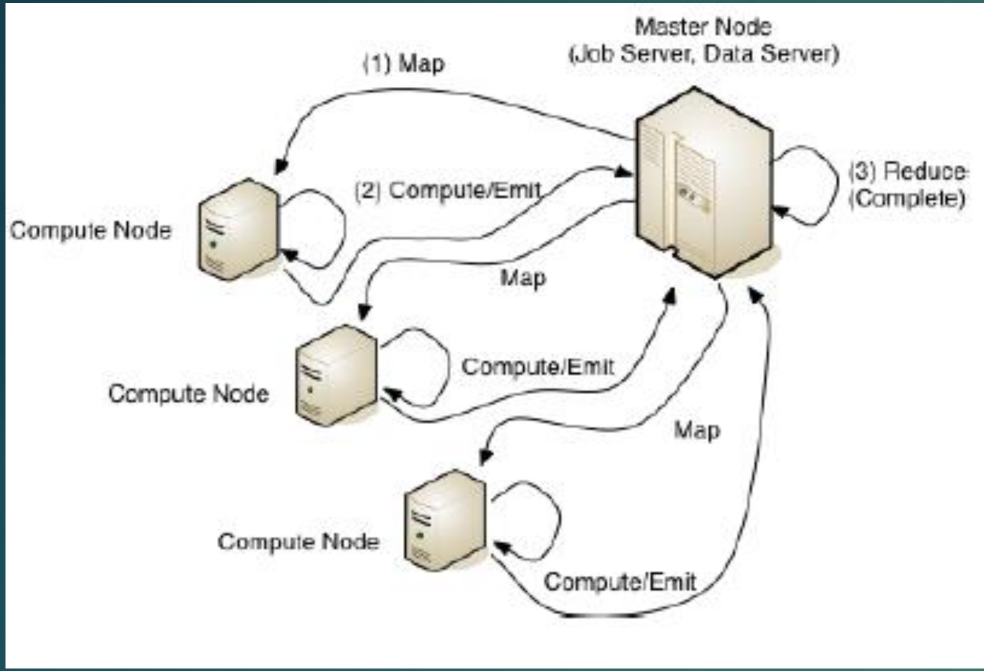
Constant increase in data volume underscored need for more and more computational power

Key Components in Proposed System

- A Server Machine – master and co-ordinator for map-reduce process
- Server side client code – used for faster and more powerful processing
- Mobile client device which implements map reduce
- BUI (Browser User Interface)

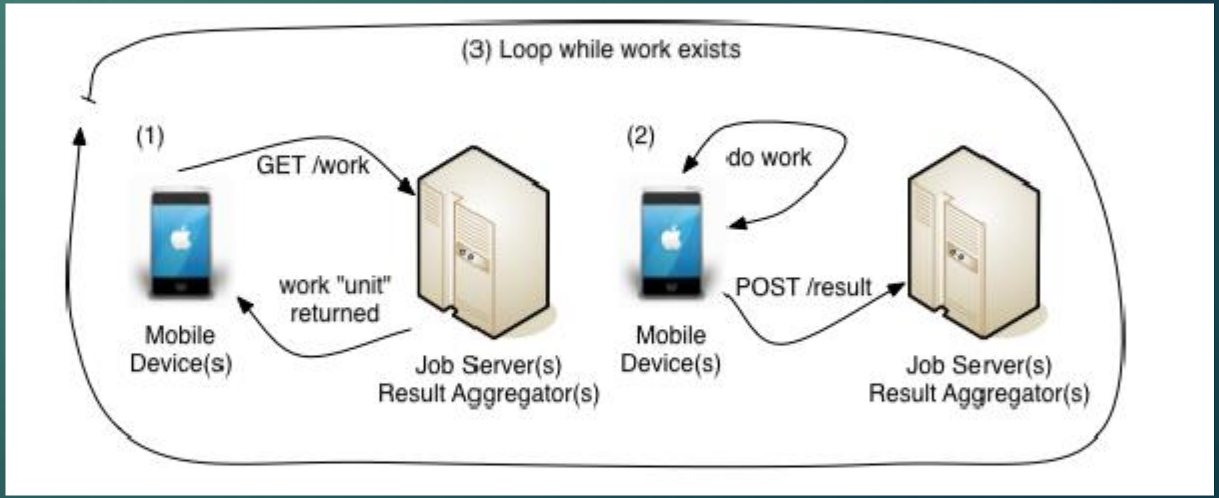


Work Flow Diagrams



High Level Map Reduce System Explaination

Work Loop



Event Driven Interruption Handling

Certain Events override the application and take control of the mobile device

- **Phone Call**
  - Application pauses during the call
  - Application is re-launched after the call
  - Computation state is saved by application
- **SMS Alert**
  - Application runs in background until the SMS is viewed
- **Calendar Event**
  - Application runs in background until the Calendar Event is viewed

End-user Participation

Two Type of Users

- **Captive**
- **Voluntary**

Experimental Setup

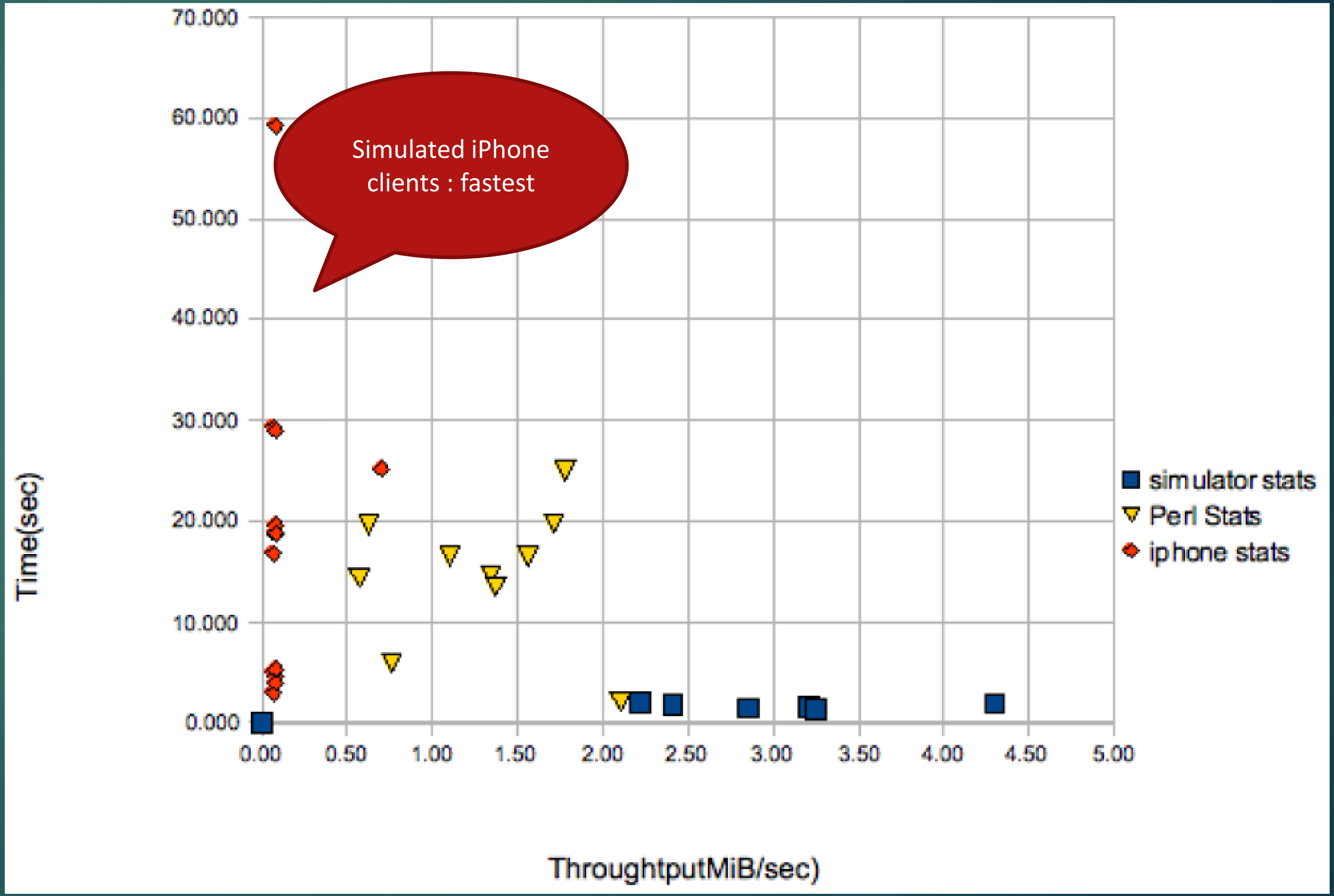
- Test devices:
  - Standard Linux server
  - iPhone
  - iPhone simulator

- Data set:
  - Overall sizes ranged from 5 MB to almost 50 MB
  - Within those data sets, each individual text document ranged from a few kilobytes up to roughly 64 kilobytes each

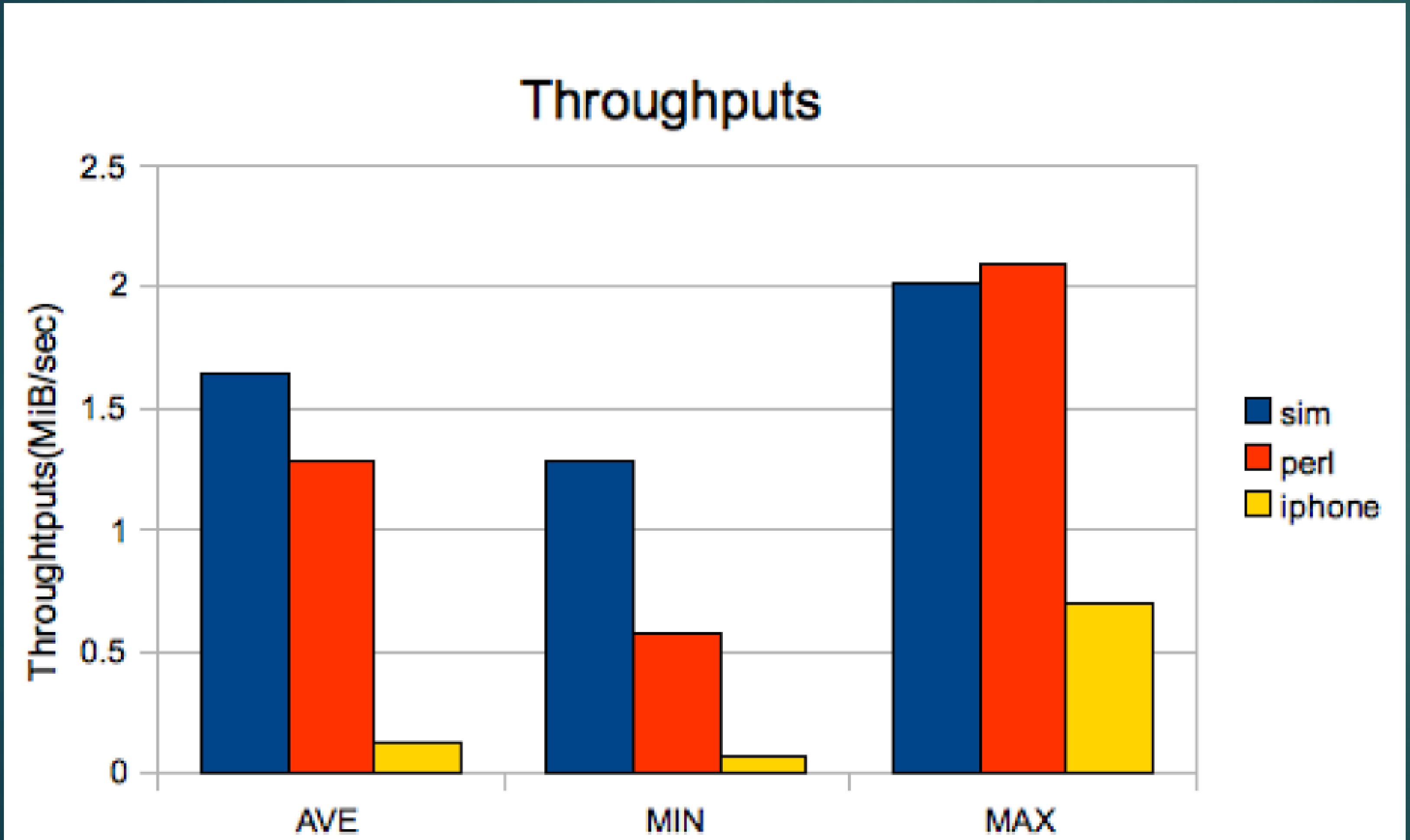
Results : Throughput per Client

- Simulated iPhone clients ran on the same machine as the server software
- Perl clients executed on remote Linux machines

- **Mixing and matching client types didn't seem to impact the contribution of any one particular client type**



Results : Variations in Throughput for different Client types



- Simulated iPhone clients : 1.64 MB/sec
  - Processed most data
- Perl clients : 1.29 MB/sec
- Real iPhone clients : 0.12 MB/sec

Results

Observation

Results consistent across a variety of data sets in terms of size and textual content

Communication

Main factor to cause processing lag

Difference in simulated and real iPhone

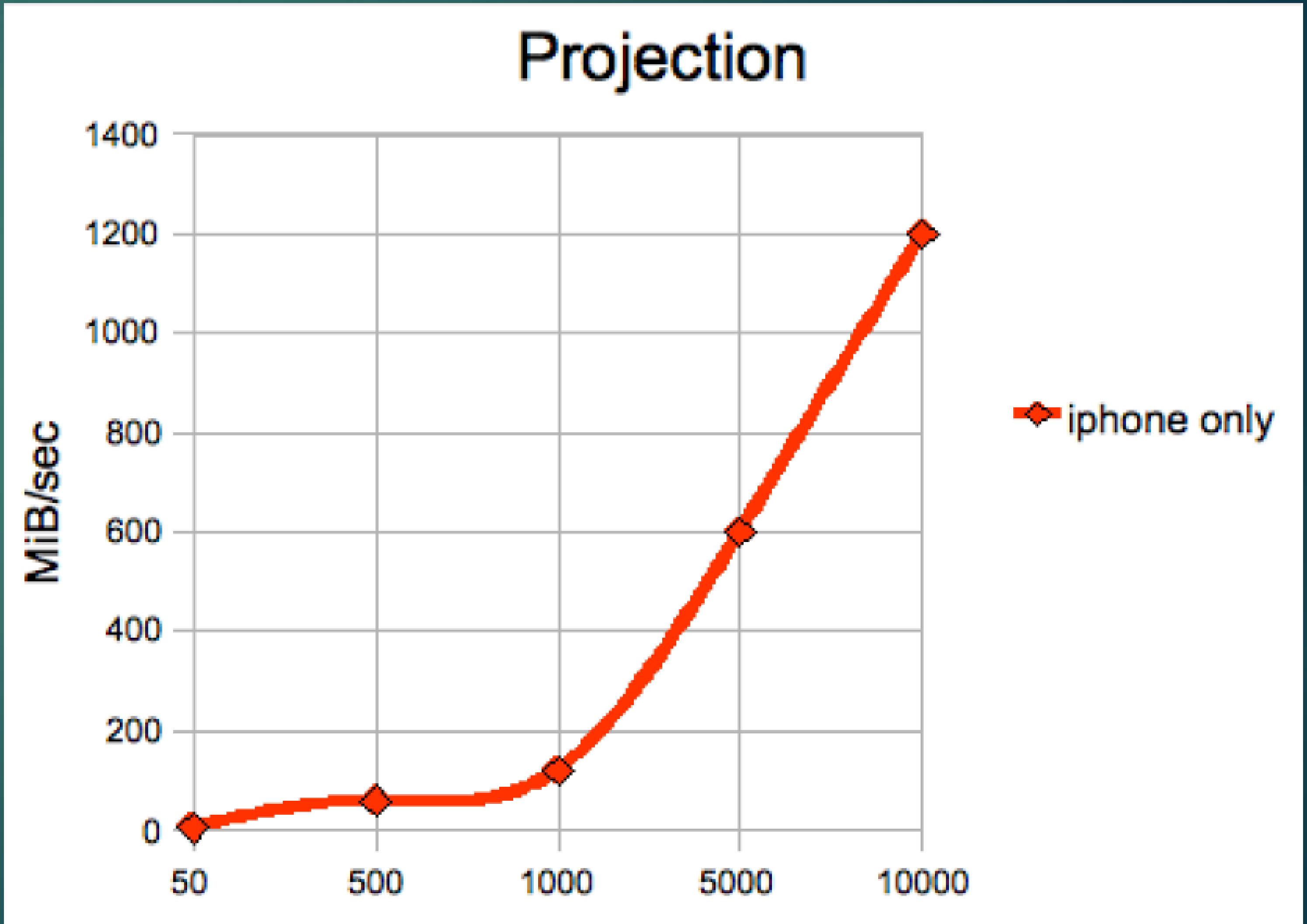
Overhead in the wireless connection and processing capabilities

Particularly useful for non-time sensitive computations

iPhone performance an order of magnitude slower than the traditional clients → considering the number of available clients, a large number of processing could be shifted to these clients

Projection : Throughput as Number of Devices Increased

- 500 mobile devices → close to 60 MB/sec of textual data
- 10000 devices →1,200 MB/sec (1.2 GB/sec!) of data
- Other components of the system would definitely start becoming bottlenecks



Scope for Optimization

Automatic Discovery

Other Client Types

Device Specific Scaling

Other Considerations

Security

Participation Incentives

Power Usage

Reference:
www.nemsausa.org
searchpp.com
community.spiceworks.com
www.digitaltrends.com
www.findandconvert.com

- Why using mobile devices for such processing is a good idea?
  - New set of mobile devices useful for large data processing
- Attempt to make MR over mobile devices Real Time
  - Scheduling for Real-Time Mobile MapReduce Systems

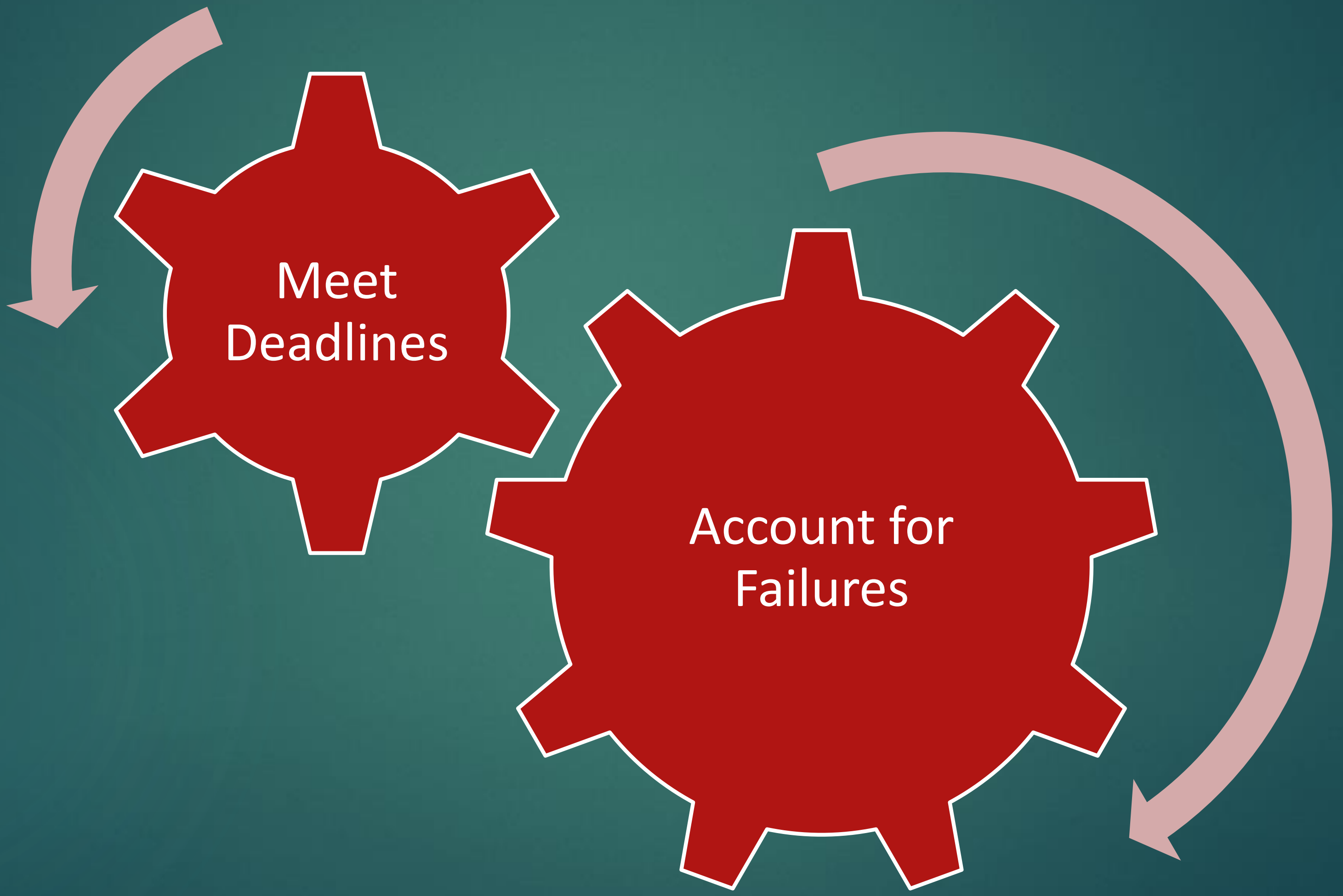
Problem Statement

- Supporting real-time applications in mobile settings is challenging due to limited resources, mobile device failures and the significant quality fluctuations of the wireless medium
- Real-Time Mobile MapReduce(MiscoRT) - proposed system - aimed at supporting the execution of distributed applications with real-time response requirements
- Effectively predicts application execution times and dynamically schedules application tasks

Challenges to be addressed

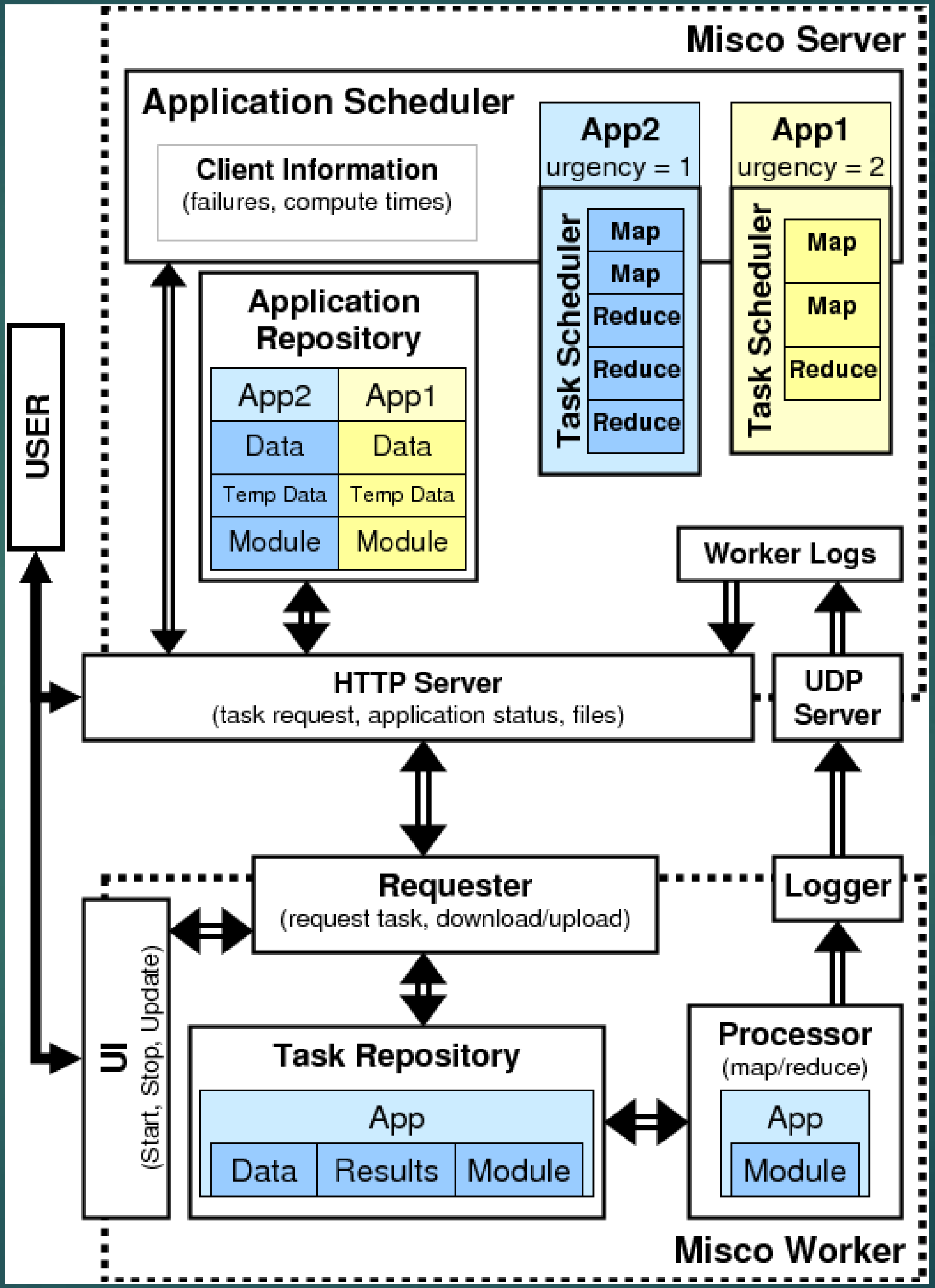
- **Application development over networks of smartphones**
  - Memory management and Application flow via new software paradigms
  - Concurrency issues
- **Application Programmability**
  - Program, develop and deploy portable applications
- **User Participation**
- **Achieving Real-Time Response**

Objectives



Misco

- MapReduce implementation that runs on mobile phones



MICRORT

- N distributed applications $A^1, A^2, ..A^N$
- M worker nodes $W^1, W^2, ...W^M$
- A^j -> consists of a number of *map tasks* (T^j_{map}) and a number of *reduce tasks* (T^j_{reduce})
- Distributed applications are triggered by the user - aperiodic and their arrival times are not known a priori
- Each application –
  - ***ready time*** r_j ***Deadline***$_j$ ***exec time***$_j$
  - ***exec time***$_j$ **--> number of map and reduce tasks, size of data, M (all are recorded)**
- *Laxity*$_j$ = Deadline - exec time
  - **Adjusted dynamically based on queuing delays and failures**
  - **Smaller → Higher Priority**
- For each task *t* of an application *Aj* compute: the *processing time* $\tau^j_{t,k}$, the time required for the task to execute locally on worker *Wk*

MICRORT

- Schedules map and reduce tasks to execute in parallel on the worker nodes
- Map or reduce
- Cannot preempt task once assigned
- Execution of tasks from different applications can interleave
- Worker only responsible for executing the current task
- Worker does not keep track of completed tasks (and from which applications)
- Server maintains this information
- System ensures independence of tasks and provision of proper data

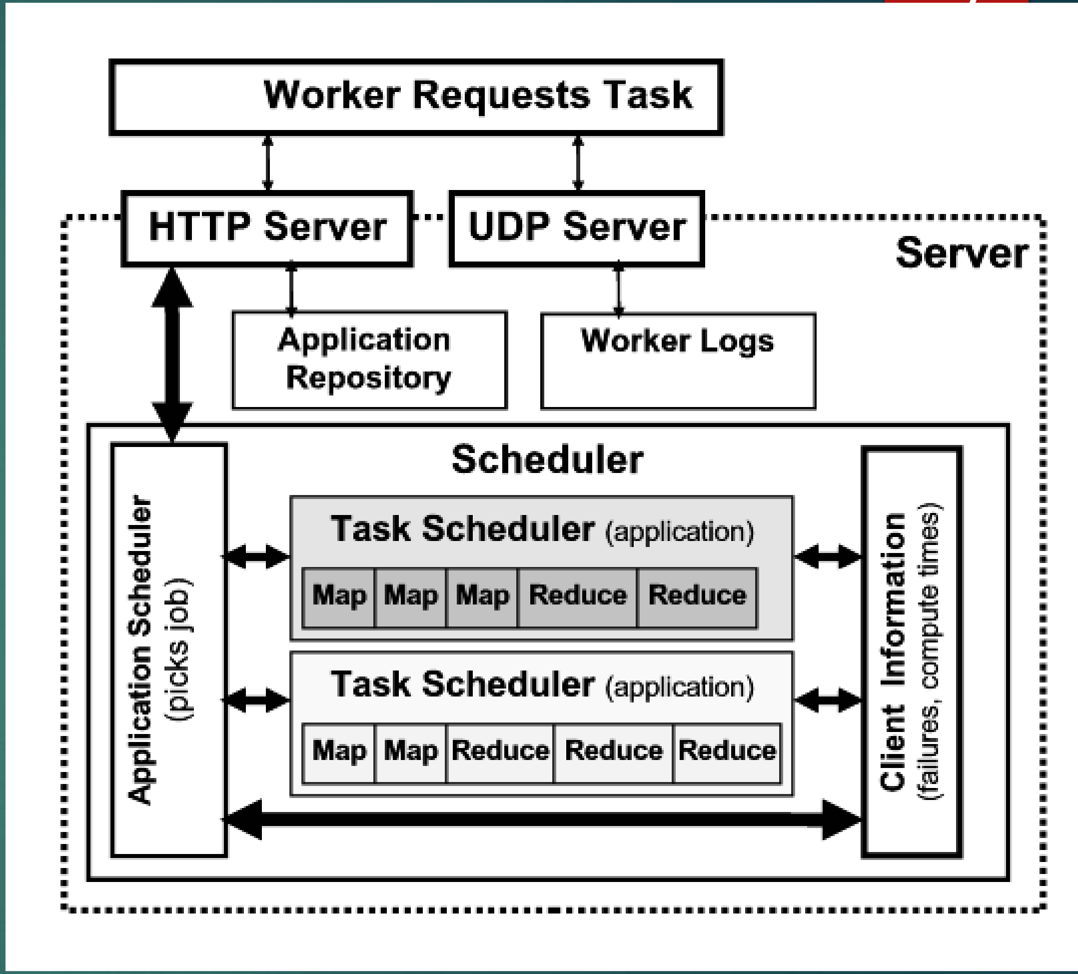
Scheduling Scheme

Application Scheduler

- determine the order of execution for the applications in the system

Task Scheduler

- ensure that all tasks of the application are scheduled for execution
- may dynamically change the number of workers allocated to the application to compensate for failures or queuing delays



- Main responsibility : To assign tasks to workers when they make requests

Failure Model : Single task, single worker

- Assumption: Failures of the worker devices follow a Poisson distribution and that failures are transient
- For application A_j and worker W_i:

λ_i - failure arrival rate for worker W_i	τ^j_i - local processing time for task of application A_j on worker W_i
μ_i - mean recovery time from a failure for worker W_i	w^j_i - expected task processing time including failures

- The expected processing time for a single task on a single worker, including failures :

$w = \tau$ *.....a successful run*

$+$

$\tau/2 * \tau\lambda/(1 - \tau\lambda)$ .....Sum of all the times wasted processing a task before failures occur

$+$

$(\mu * \tau\lambda)/(1 - \tau\lambda)$ .....Sum of all the downtime in order for the worker to recover from failures

Failure Model : Multiple Tasks, Multiple Nodes

- For application A_j and worker W_i: Consider T tasks belonging to same application

λ_i - failure arrival rate for worker W_i	τ^j_i - local processing time for task of application A_j on worker W_i
μ_i - mean recovery time from a failure for worker W_i	w^j_i - expected task processing time including failures

- The total execution time for all T tasks of application A_j

  = maximum (individual processing times for each worker)

- Since all workers are either processing a task or in a failure state, we can model this by considering a equal-time workload for each worker
- For the workers to finish their tasks at the same time, the number of tasks ρ_i assigned to worker Wi $(1 \leq i \leq M)$ is:

$$\rho_i = \lceil \frac{1/w_i}{\sum_{k \in M} 1/w_k} * T \rceil$$

- Expected execution time $= max_{i \in M}(\rho_i * w_i)$

Application Scheduler

- Least-laxity scheduler
- ***Laxity$_j$ = Deadline$_j$ – current time – exec time$_j$***
- Schedule is driven by both the timing requirements of the applications and node failures
- *Slower processing → decreased laxity → higher priority*

MiscoRT Application Scheduler

Input: Set of applications A in system
for all Application A_j in A **do**
  calculate $Laxity_j$ of A_j
Order A by $Laxity_j$
Task ← TaskScheduler(A_j with smallest $Laxity_j$)
return Task

Task Scheduler

- Ensure all tasks are scheduled for execution
- Dynamically change workers allotted to each task to compensate for queuing delays and failures
- 3 step process:

MiscoRT Task Scheduler

Input: worker W_k requests a task, job A_j

step 1. **if** unassigned task $T_i^j \in A_j$ **then return** T_i^j

step 2. **if** failed task $T_i^j \in A_j$ **then return** T_i^j

step 3. $T_i^j \leftarrow$ slowest task in A_j

  if T_i^j will complete after $deadline_j$

    and T_i^j will complete on W_k before $deadline_j$ **then**

      return T_i^j

Experimental Setup

- **Mobile Clients:**
  - 30 Nokia N95 8GB smart-phones
  - ARM11 dual CPUs at 332 Mhz
  - 90 MB of main memory and 8 GB of local storage
  - Supports wireless 802.11b/g networks, bluetooth and cellular 3g networks
- **Server:**
  - A commodity computer
  - Pentium-4 2Ghz CPU
  - 640 MB of main memory.
- **Communication:**
  - The server has a wired 100 MBit connection to a Linksys WRT54G2 802.11g router.
  - All of the phones are connected via 802.11g to this router.

Application Specs and Baseline Case

- 11 Applications – 8 with 100kB input and 3 with 1MB input
  - 5 applications have tight deadlines
  - 2 applications have medium deadlines
  - 3 applications have loose deadlines
- Baseline Comparison – **Earliest Deadline First**
- Parameters:
  - Miss Ratio
  - End to end time

Results

- Uniform distribution of worker failures

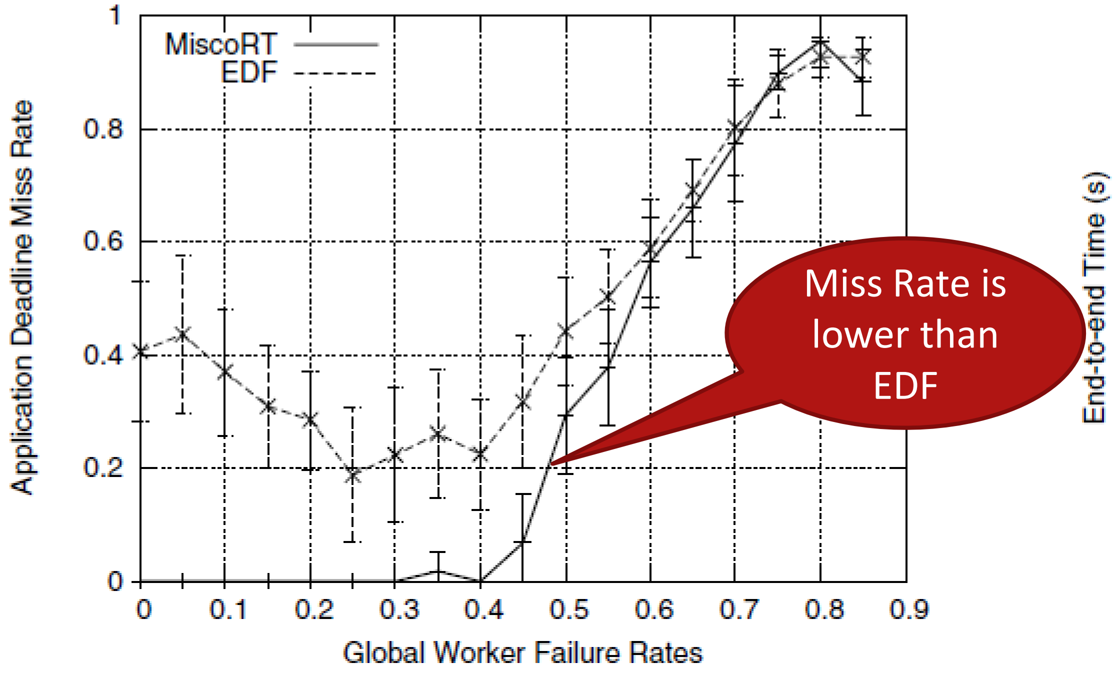


Figure 4: Application miss rate for MiscoRT compared to EDF with uniform distribution of worker failures.

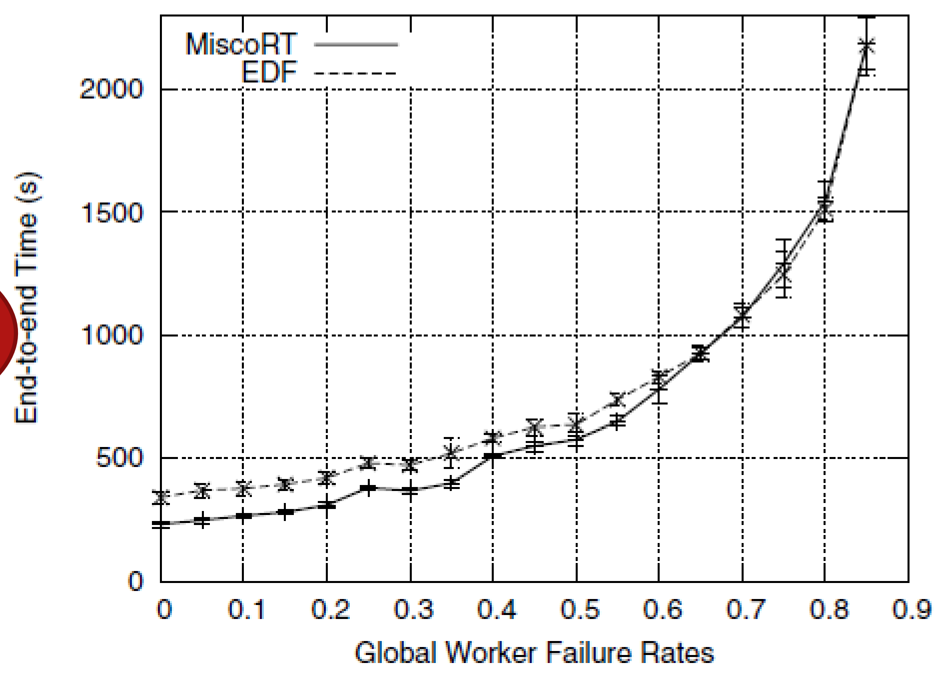


Figure 5: End-to-end times for MiscoRT compared to EDF with uniform distribution of worker failures.

Results

- Lognormal distribution of worker failures

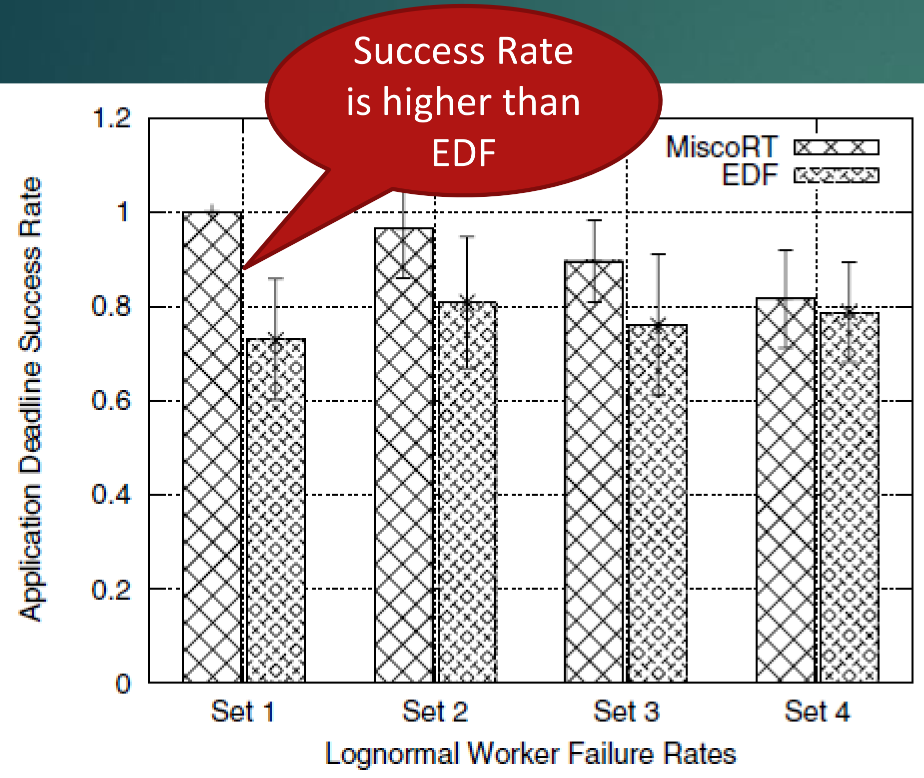


Figure 6: Application success rates for MiscoRT compared to EDF with lognormal distribution of worker failures.

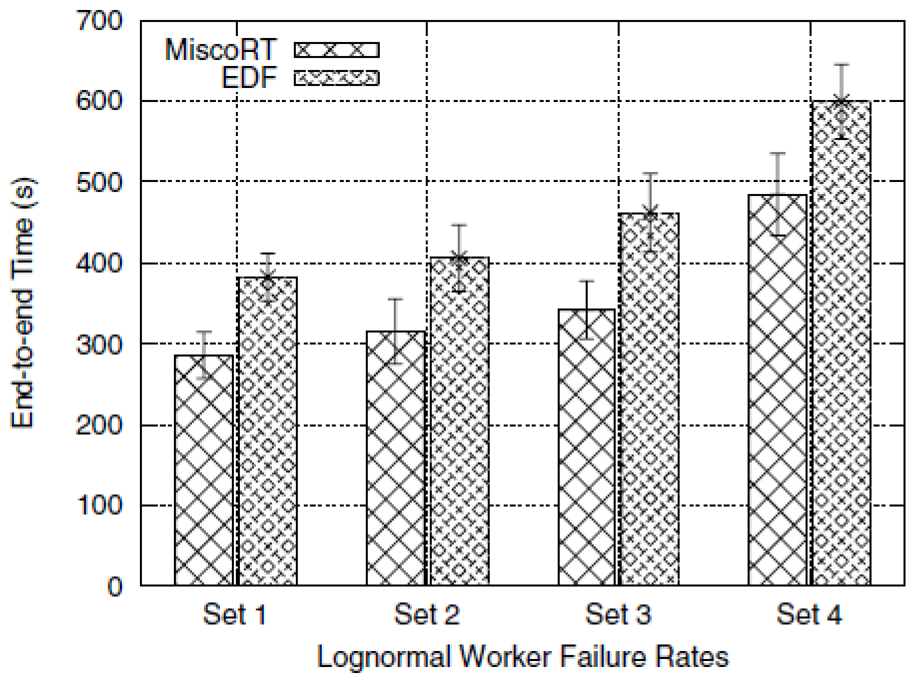


Figure 7: End-to-end times for MiscoRT compared to EDF with lognormal distribution of worker failures.

Comparison with different Task Schedulers

Random Task Scheduler
- Selects tasks at random
- Very low overhead
- Wastes computational resources

Sequential Task Scheduler
- Picks Tasks sequentially, hence low overhead
- Does not consider worker failures
- Avoids duplicate assignment

Modified Hadoop Task Scheduler
- FIFO based task scheduler
- Constant worker feedback about their progress

Results

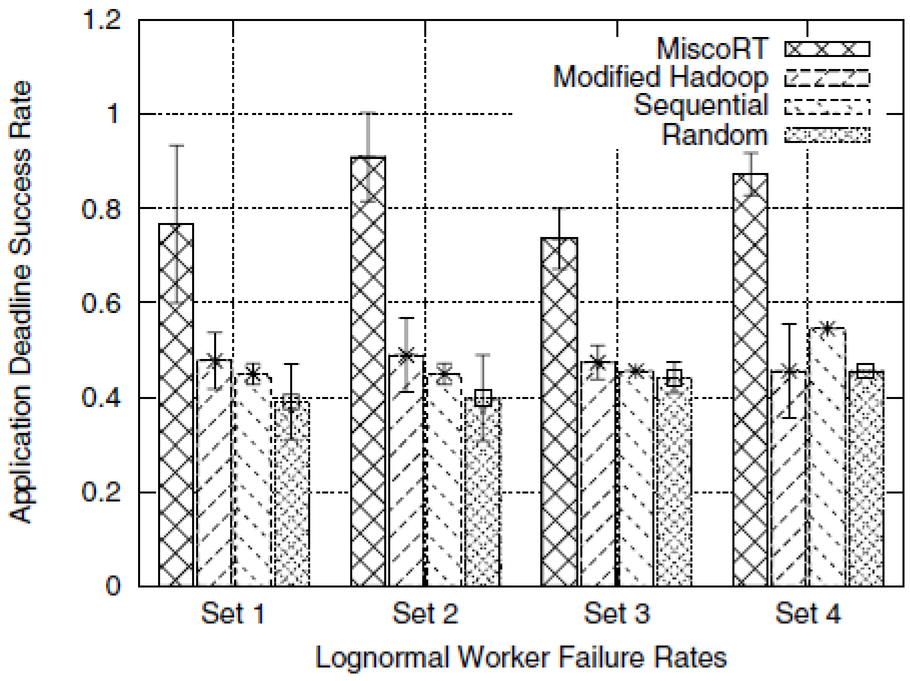


Figure 8: Application success rates for MiscoRT compared to other task schedulers. Each task scheduler was paired with the MiscoRT application scheduler

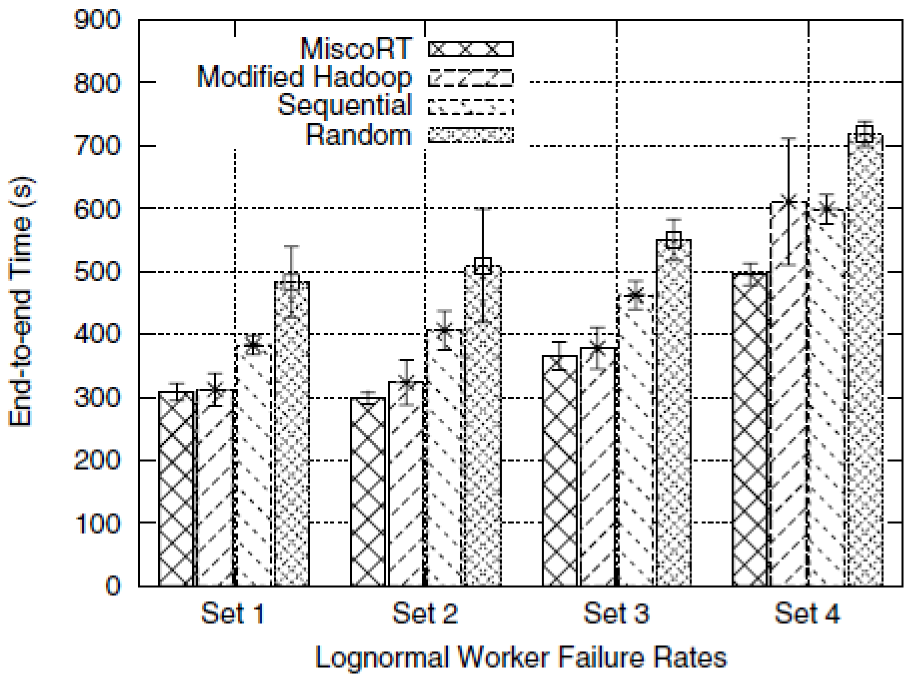


Figure 9: End-to-end times for MiscoRT compared to other task schedulers. Each task scheduler was paired with the MiscoRT application scheduler

Validation

- Compare predicted execution time with actual execution time
- 1 application with 73 tasks
- Assume all workers fail with same rate

Predictions are very accurate even at high failure rates

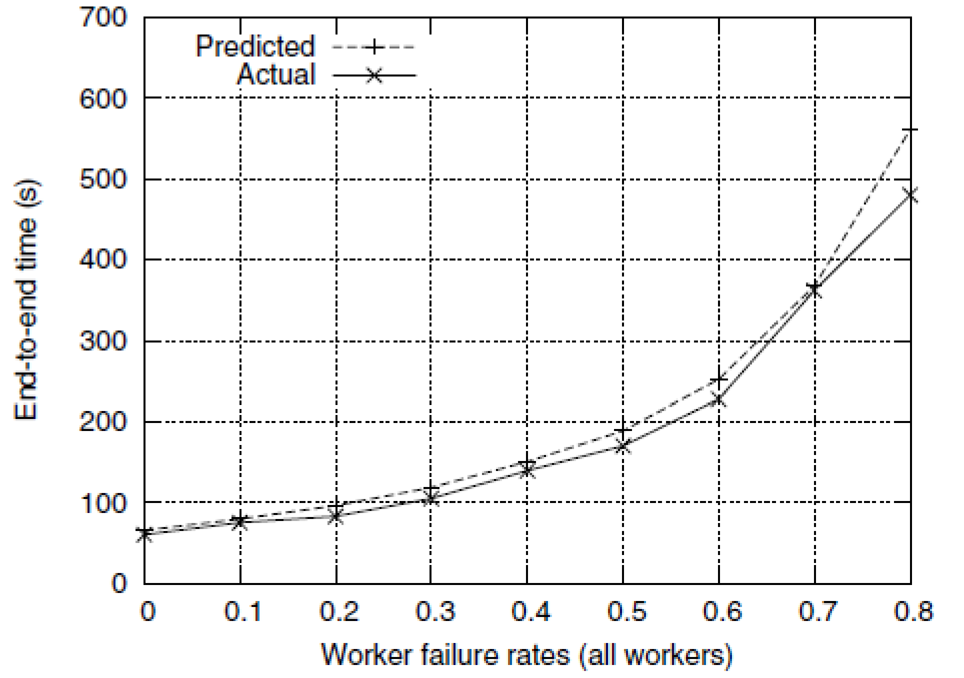


Figure 10: Model validation over various worker failure rates.

Scalability

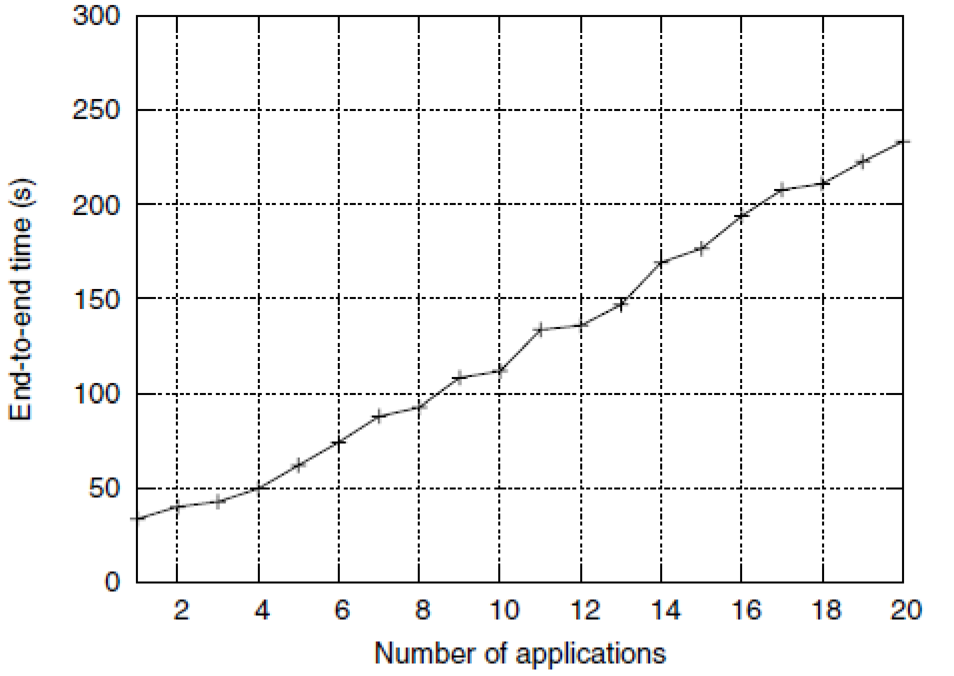


Figure 11: End-to-end time as number of applications are varied.

- Number of applications is increased linearly
- Failure rate is set to 0
- Processing power is fixed

End-to-end time increases linearly with increase in applications

Deadline Sensitivity

- Deadlines are made tighter by 20% for each test
- Failure rate is kept constant at 20%
- Comparison of Miss Rates of EDF and proposed Scheduler

EDF has more misses than proposed scheduler

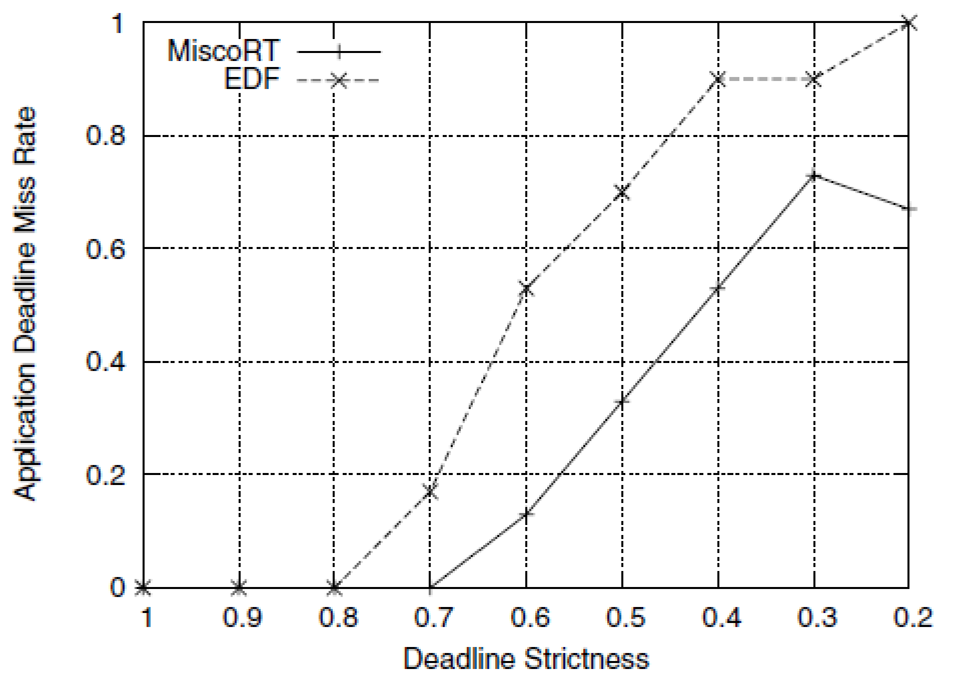


Figure 12: Application miss rates as a function of deadline strictness.

Overhead and Resource Usage

CPU, Memory and Power Consumption is measured using NOKIA Energy Profiler

- **CPU**
  - Task dependent and also takes into consideration other applications running on phone
  - Application gladly uses all processing power available to it
- **Memory**
  - Application needs only 800kB Memory
  - Scheduler does not introduce any overhead (only 150 lines of code)
  - Almost 90MB Memory free
- **Power Usage**
  - Processing data requires 0.7 watts
  - Network access requires 1.6 watts
  - It is much more effective to process data locally than to send it over network

Conclusion

- Map-reduce framework can be implemented on Mobile Devices to utilize their huge potential of performing highly distributed compute intensive applications
- Failure is not an exception, but a Norm in such a system. Deadlines should be met even in the face of Failures
- A scheduler is proposed that

  (1) performs effectively, even under failures,

  (2) has low overhead,

  (3) consistently outperforms its competitors

Drawbacks

First paper :

- No information about Versions and configuration details

Second Paper :

- Did not conducts tests on network performance

Thank You