High Performance Computing and Networking Center

Efficient Coscheduling Model and Implementation for Beowulf Cluster

Somsak Sriprayoonsakul <ssy@hpcnc.cpe.ku.ac.th>,

Asst. Prof. Dr. Putchong Uthayopas <pu@ku.ac.th>

High Performance Computing and Networking Center,

Faculty of ENGineering,

Agenda

- *Introduction*
- *Problem*
- *Research Goal*
- *Related Work*
- *Proposed ECM/EDM model*
- *Experimental Evaluation*
- *Conclusion and Future Work*

Introduction

- Cluster computing system is now an important platform for high performance computing
  - Cost effective, scalability, performance
- High performance applications usually developed using Parallel Programming
  - MPI (Message Passing Interface)

Problems

- Performance of parallel program in Beowulf clusters environment depend on
  - The minimization of the impact from other tasks sharing the system
  - The good synchronization among tasks in parallel program that allows the speedup of message transmission
- An efficient form of synchronized scheduling is needed

Coscheduling : Introduction

- Traditional System leave local scheduler to schedule its job in each node
  - Performance drawback because of non-synchronized scheduling
  - Less utilization
- Coscheduling
  - Synchronized scheduling technique that manage multiple processes on a set of nodes together to allows more efficient execution

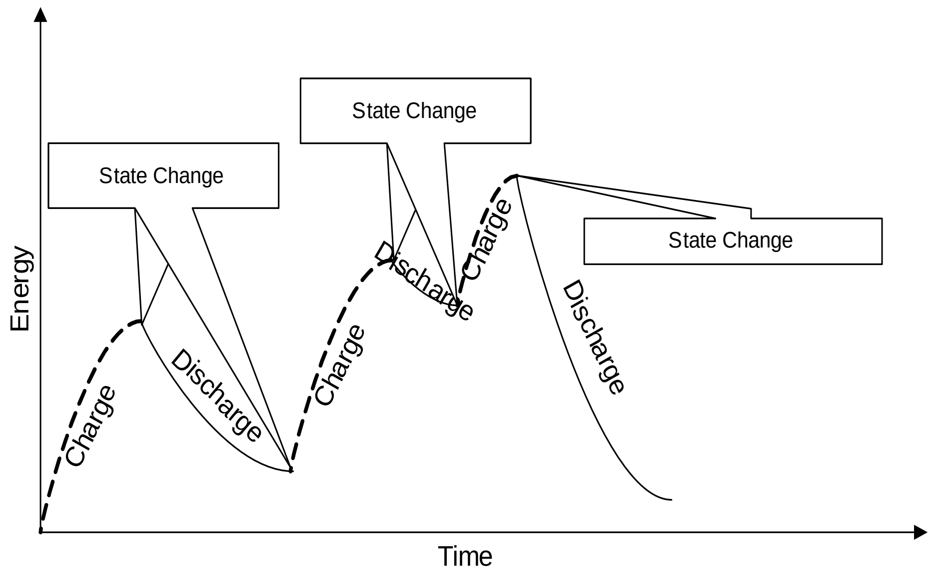
Research Goal

- Propose a new and efficient co-scheduling method.
- Develop a coscheduling implementation for parallel applications running on Beowulf Clusters
  - All pass work make the decision based on immediate data in the system
  - No algorithm use the accumulated behavior of the transmission to build an effective prediction model for task behavior
    - Past statistics
    - Current system statistics

Energy Model for Implicit Coscheduling

Energy Model for Implicit Coscheduling

- Based on Predictive Coscheduling. Priority calculation is based on the idea of “Energy”
- Concept
  - Process charge/discharge “energy” while it executes
  - The amount of charged/discharged energy is calculated by current process parameter.
  - Calculated energy is then saved for later calculation
  - The charging and discharging state changes when communication state changes



- Charge/Discharge rate is calculated from process statistics
  - Communication Frequency
  - Message Size
  - Amount of running process in the system

Energy Charging Model (ECM)

- The energy is charged while the process is in non-communication state.
- The energy is then convert to the process priority value. Coscheduler will adjust the process priority accordingly.
- Coscheduler adjust the priority only when the process is in communication state. The priority is leave as is while it computes
- The charging algorithm is

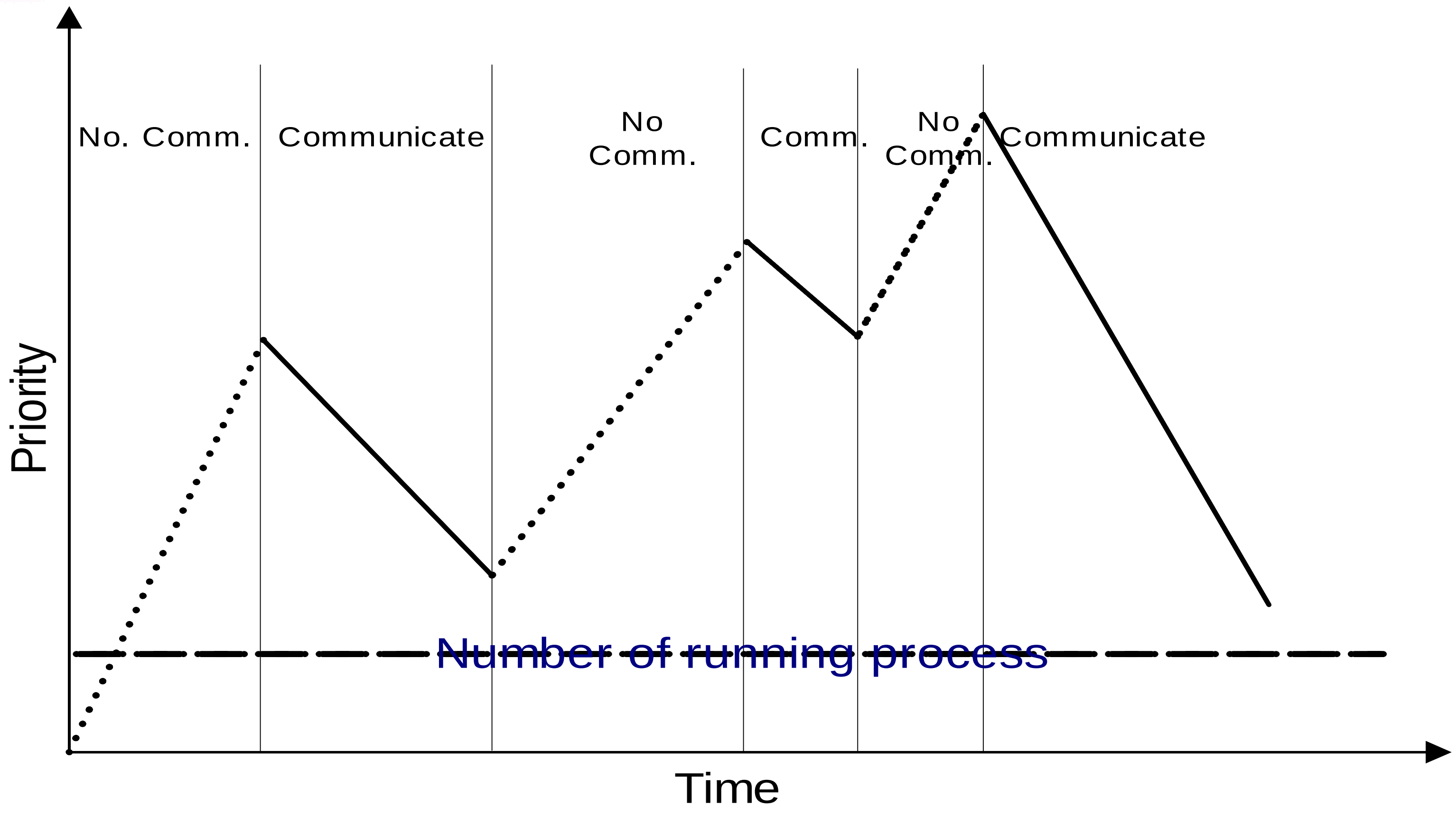
$$P = \alpha f t + P_{last} \qquad (1)$$

- The discharging algorithm is

$$P = \begin{cases} -\beta \dfrac{t}{m} + P_{last} & \text{if } P > R; \\ R & \text{if } P \leq R. \end{cases} \qquad (2)$$

- Where
  - P = priority
  - P_{last} = Last priority value
  - f = communication frequency
  - t = time elapsed since the execution started
  - m = message size
  - R = number of running process
  - other value are constant value

Energy Charging Model (ECM)



- Energy Charging Model give the fair-share over all job in the system
- The priority will gradually drops when communication, thus prevent starvation problem
- The job with more frequent and larger message size would have lower discharge rate and higher charge rate
- The priority will not fall below the number of running process, thus guarantee priority boost while communication

Energy Discharging Model (EDM)

- As oppose to ECM, EDM reverse the behavior of ECM thus restrain most characteristics of ECM
- The priority will gradually rise while the process communicates
- The charging algorithm is

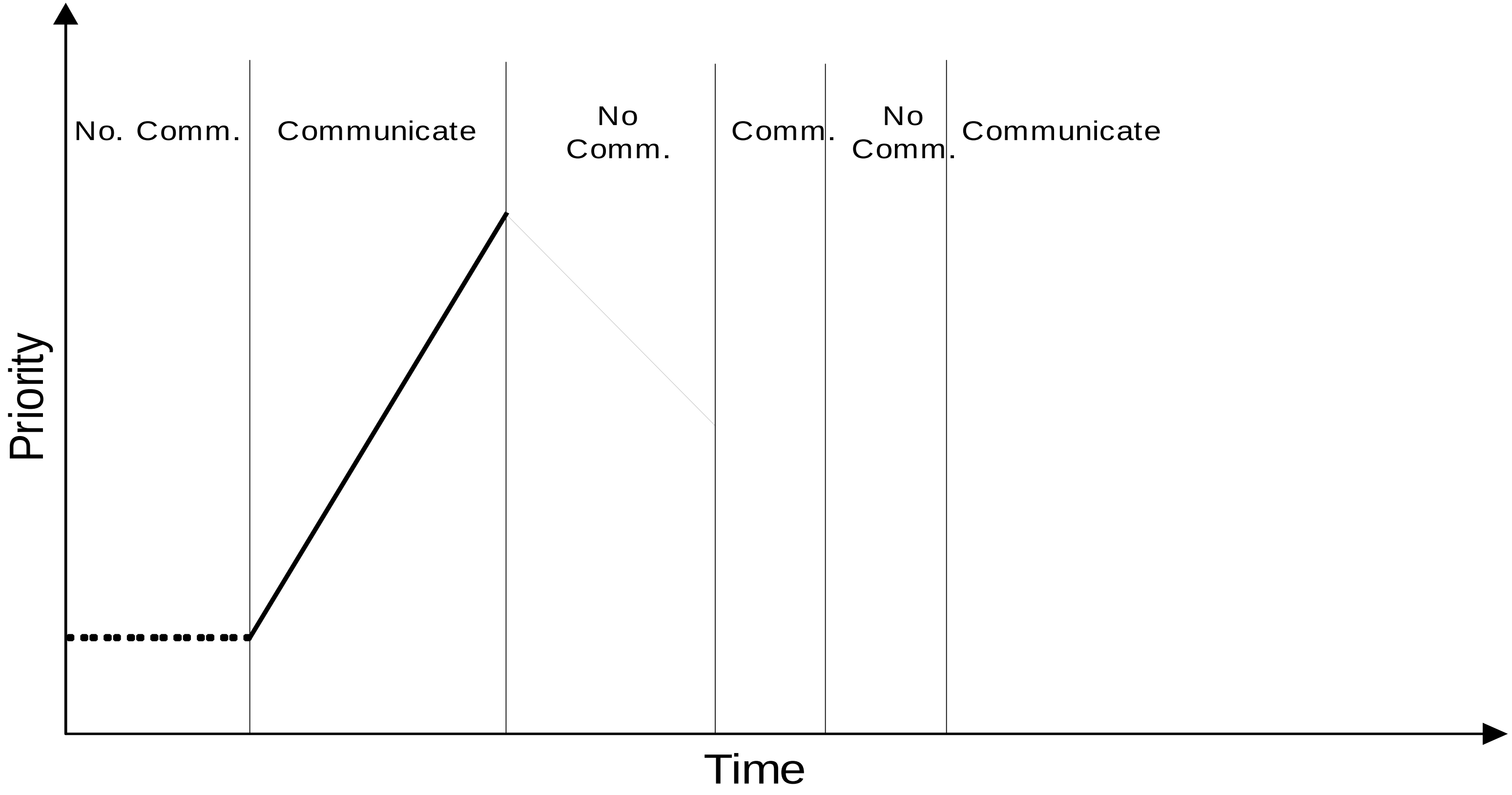
$$P = \begin{cases} -\gamma \dfrac{t}{f} + P_{last} & \text{if } P > R; \\ R & \text{if } P \le R. \end{cases} \qquad (3)$$

- The discharging algorithm is

$$P = \varepsilon m t + P_{last} \qquad (4)$$

- Where
  - P = priority
  - P_{last} = Last priority value
  - f = communication frequency
  - t = time elapsed since the execution started
  - m = message size
  - R = number of running process
  - other value are constant value

Energy Discharging Model (EDM)

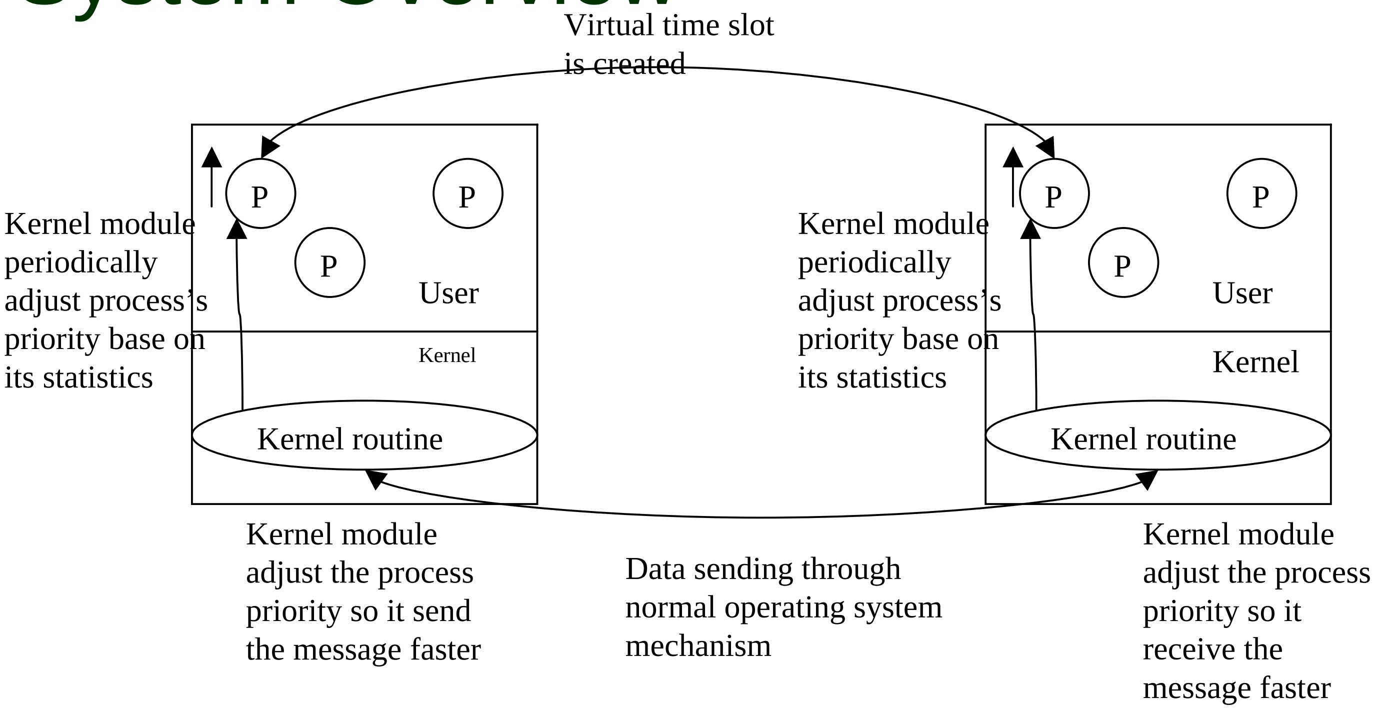


Implementation Details

- Implemented in kernel-level as Linux Kernel Module (LKM)
  - kernel version 2.4.19 (the latest at the time)
  - Using Linux timer mechanism to periodically inspect the kernel task queue and adjust the value of each task_struct
  - User need to tell the system which process to do the coscheduling by using command line.
  - _exit system call is trapped to ensure that all internal variable is cleared when process exit

Implementation Details

- System Overview



Experimental Result

- The experimental setup is
  - 4 nodes Beowulf Cluster, each node is Athlon 700 MHz with 128MB ram
  - Fast Ethernet Network
  - RedHat Linux 7.3 with Linux Kernel 2.4.19
  - MPICH 1.2.4

Testing condition

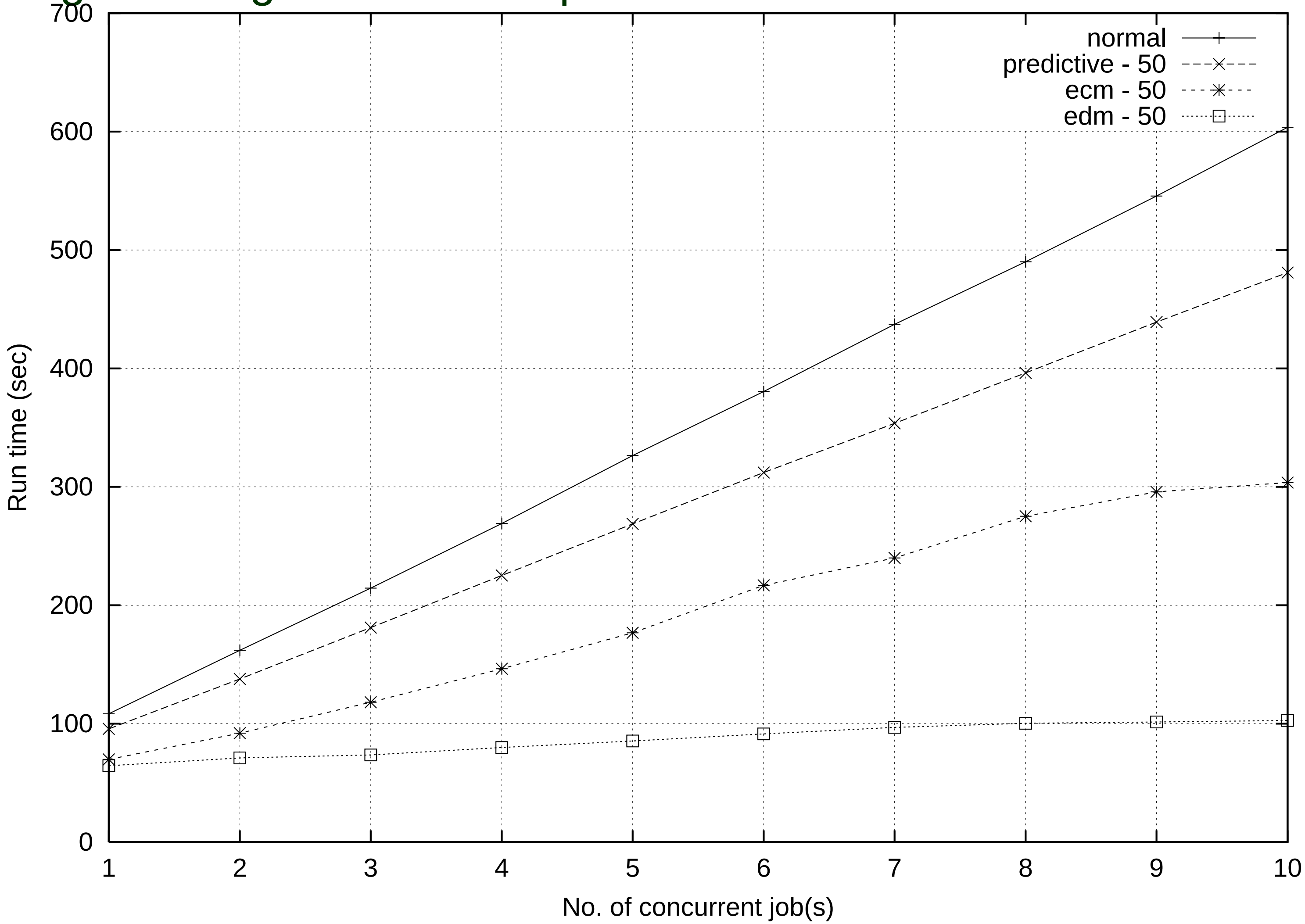
- Using NAS Parallel Benchmark version 2.3 as the test application. Three benchmarks is selected
  - IS and LU represents fine-grain parallel program
  - MG represents coarse-grain parallel program
- infcalc, a program which do infinite floating-point calculation
- All test is running with normal scheduler, Predictive Coscheduling, ECM, and EDM
- All result is the average from 5 runs
- **NOTE**: The number of application instants is limited due to memory constraint. All application is running with no memory trashing

Sequential Workload Experimental

- IS, MG, and LU is running against 1-10 instants of infcalc
- Illustrates the performance of coscheduling against multiple sequential workload

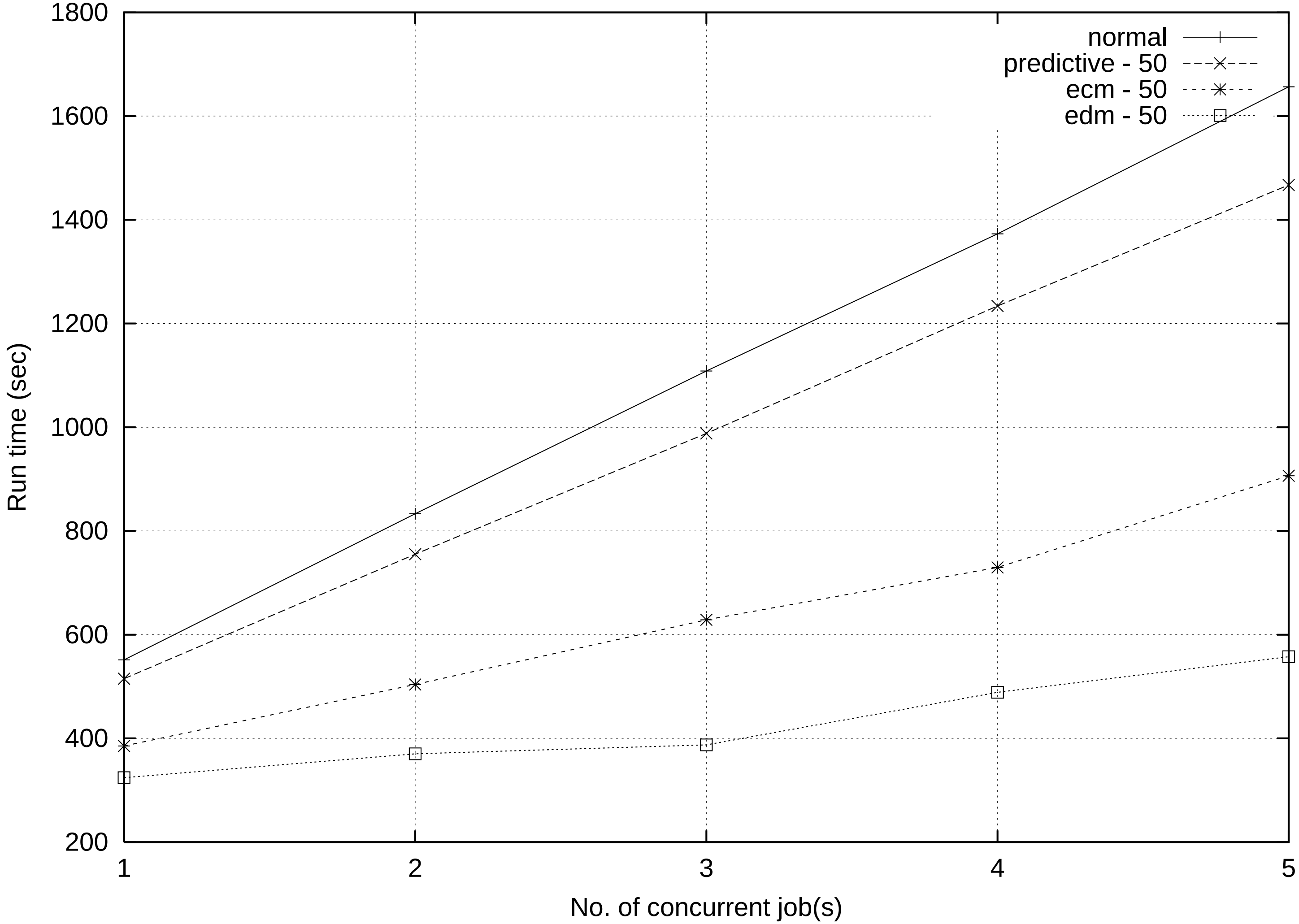
Runtime of parallel application against sequential workload

- Single MG against 1-10 sequential workload



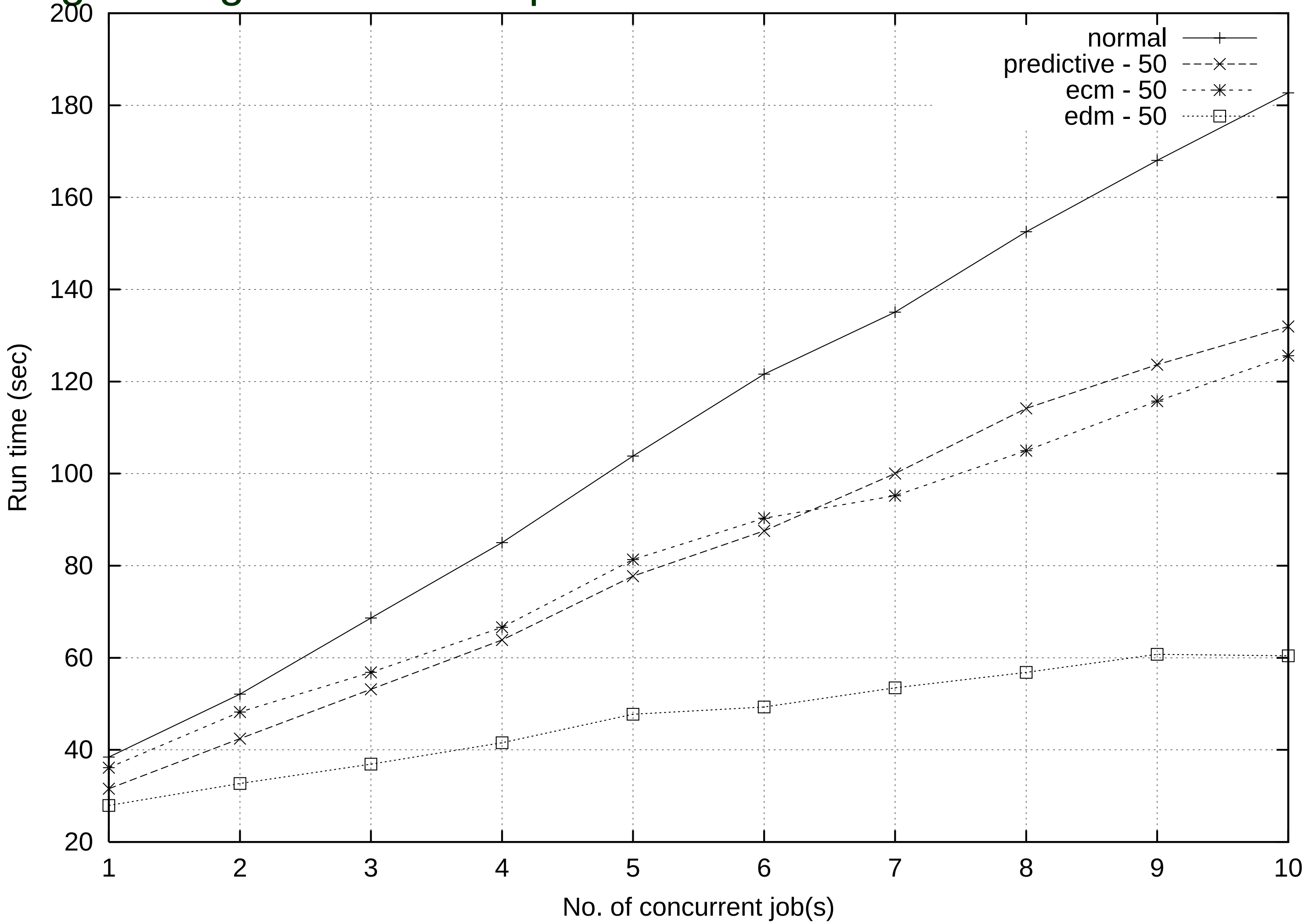
Runtime of parallel application against sequential workload

- Single LU against 1-10 sequential workload



Runtime of parallel application against sequential workload

- Single IS against 1-10 sequential workload

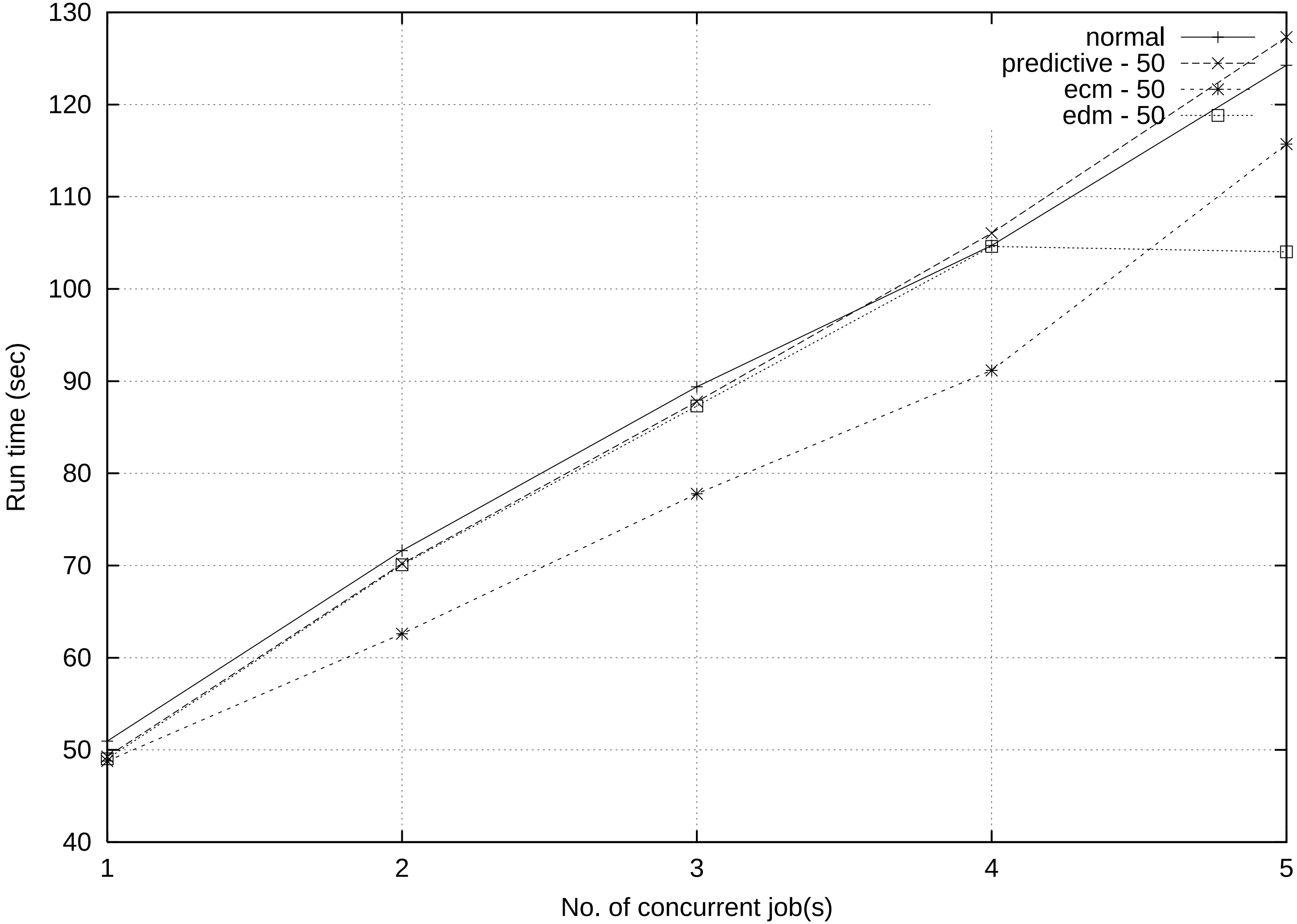


Runtime of multiple parallel application

- IS, MG, and LU is running against each other
- Illustrates the performance of coscheduling when running multiple instants of parallel program

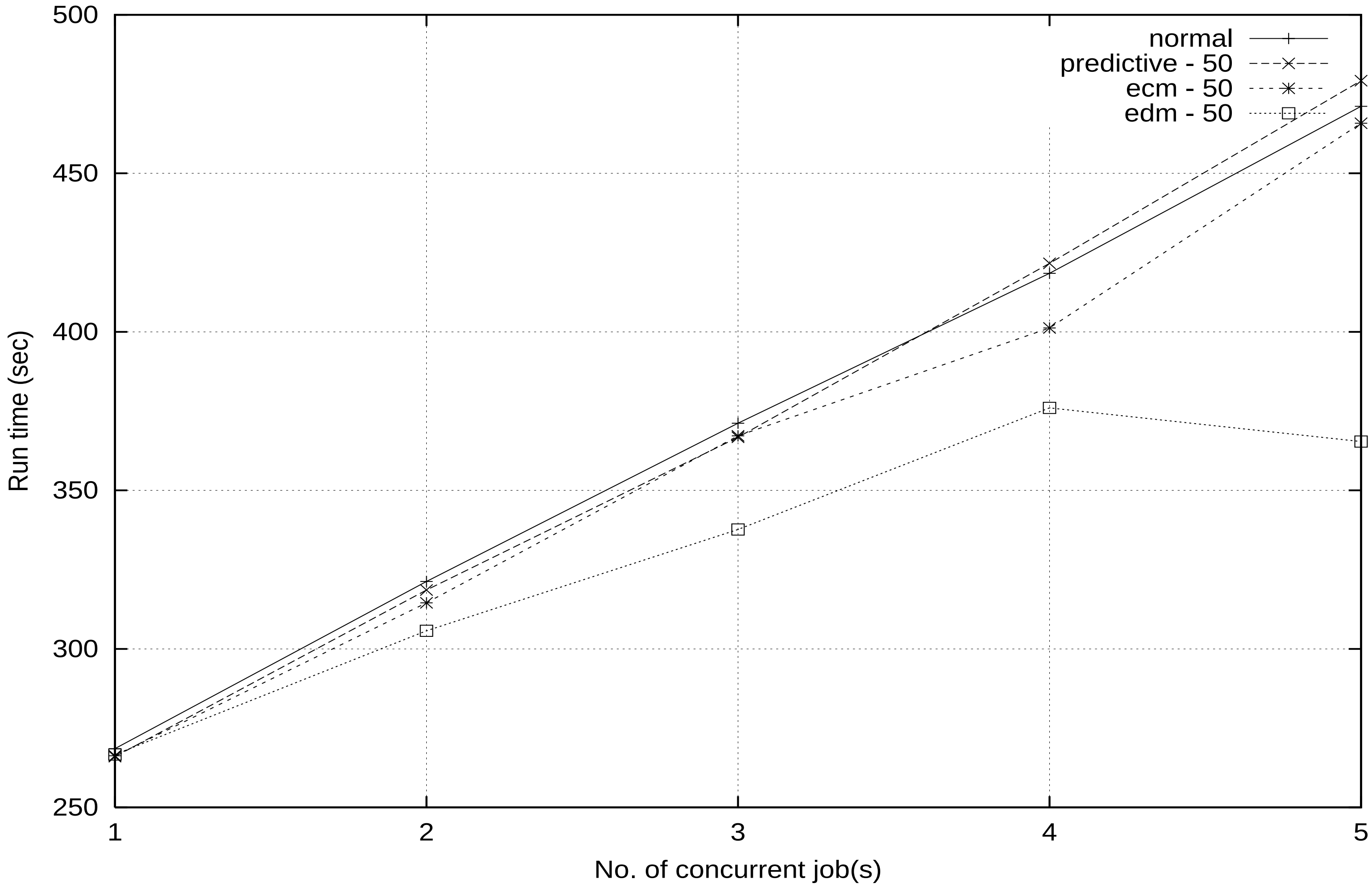
Runtime of multiple parallel application

- Single MG against 1-5 IS



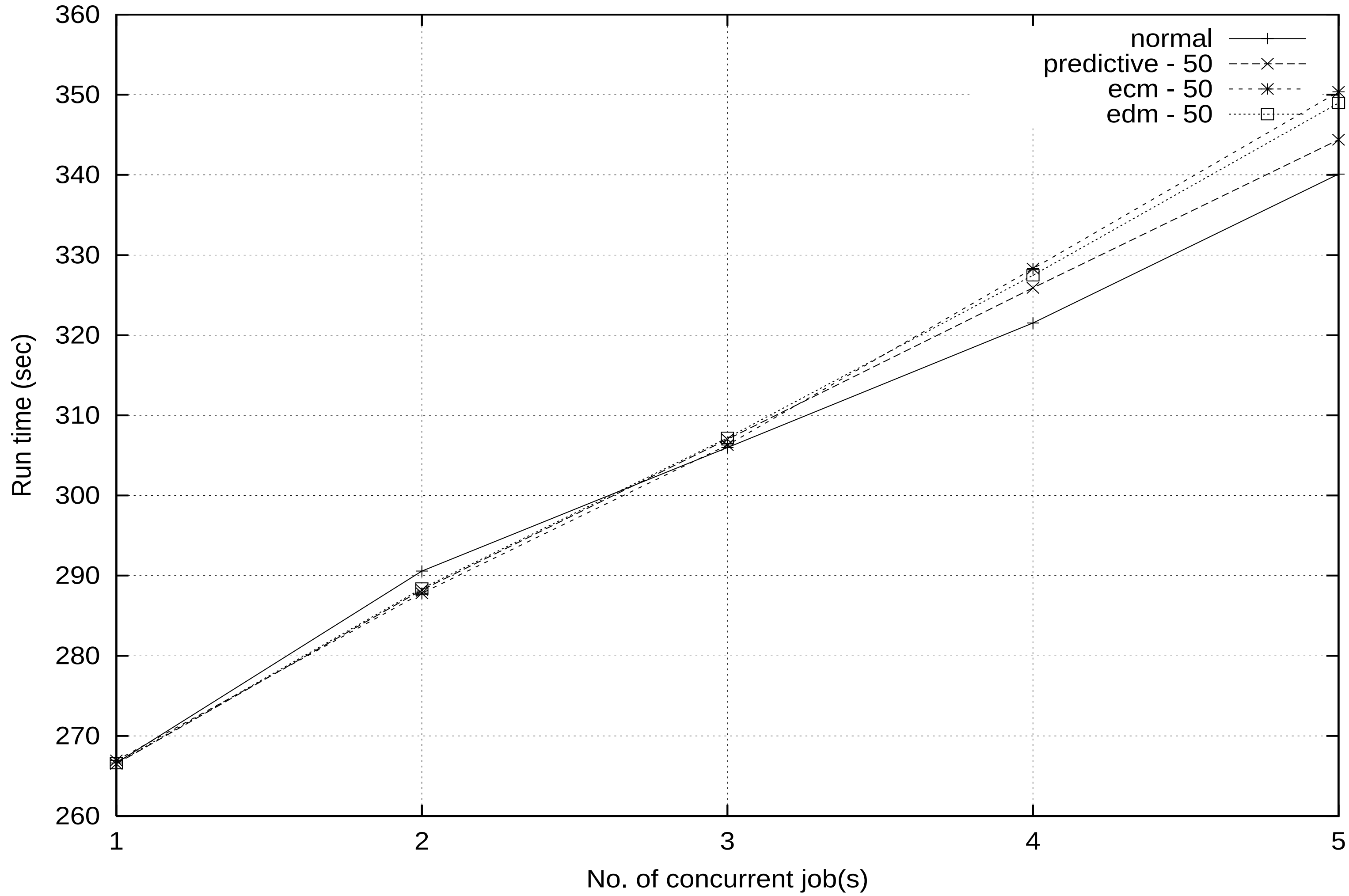
Runtime of multiple parallel application

- Single LU against 1-5 MG



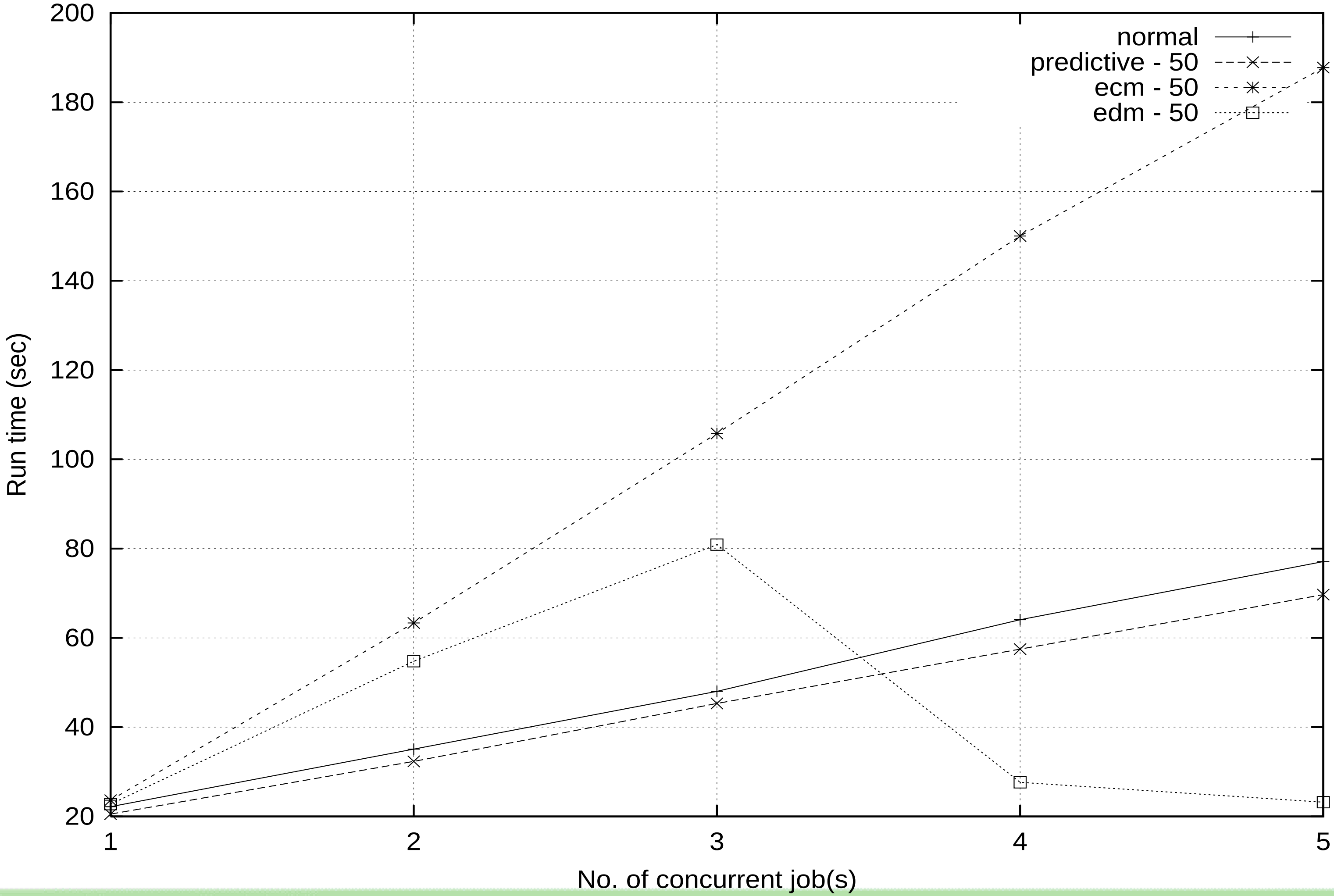
Runtime of multiple parallel application

- Single LU against 1-5 IS



Runtime of multiple parallel application

- Single IS against 1-5 MG

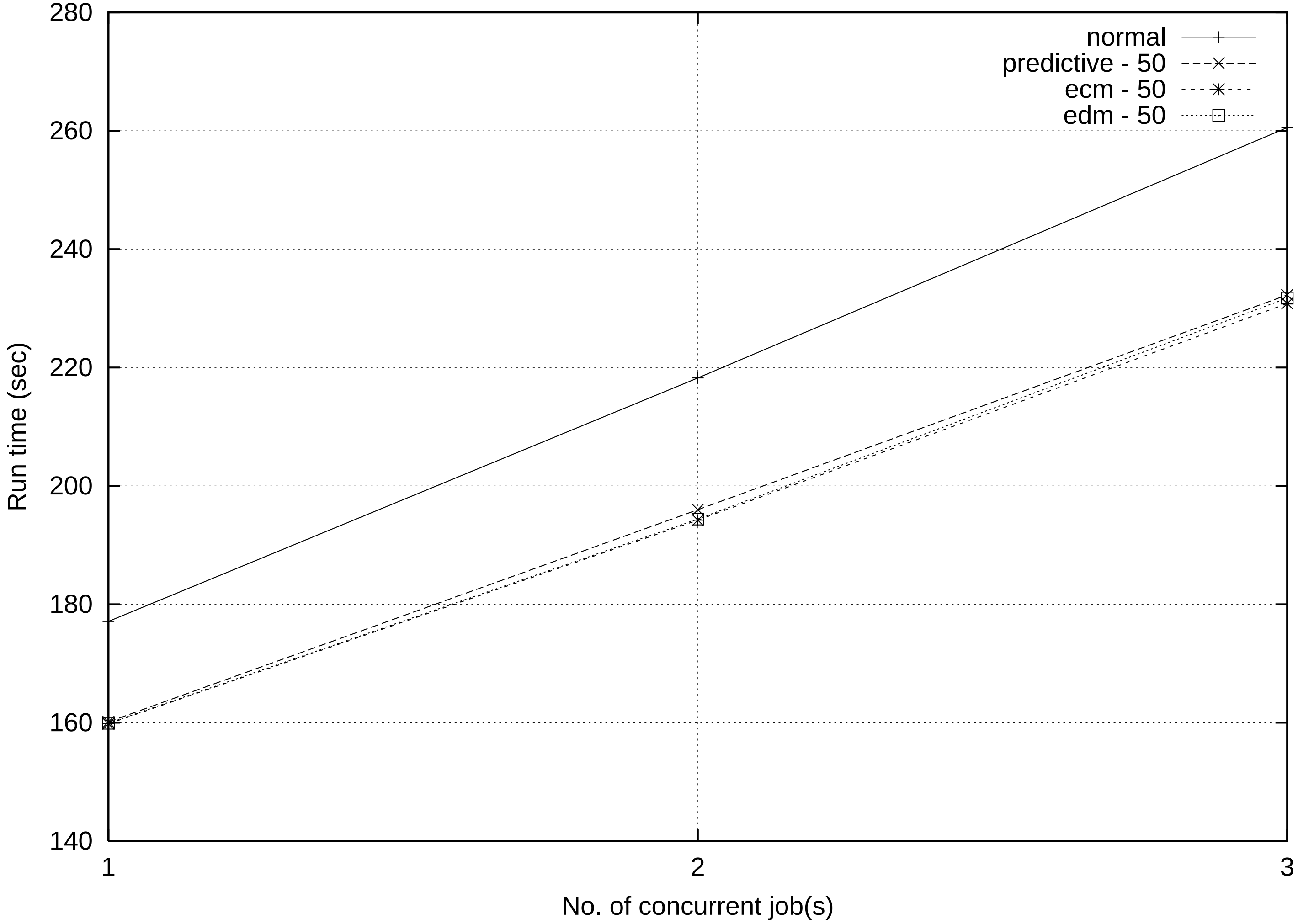


Impact of Coscheduling to Sequential task

- A sequential program is running against 1-3 instants of MG, IS, and LU
- Illustrates impact on run-time of coscheduling to sequential task

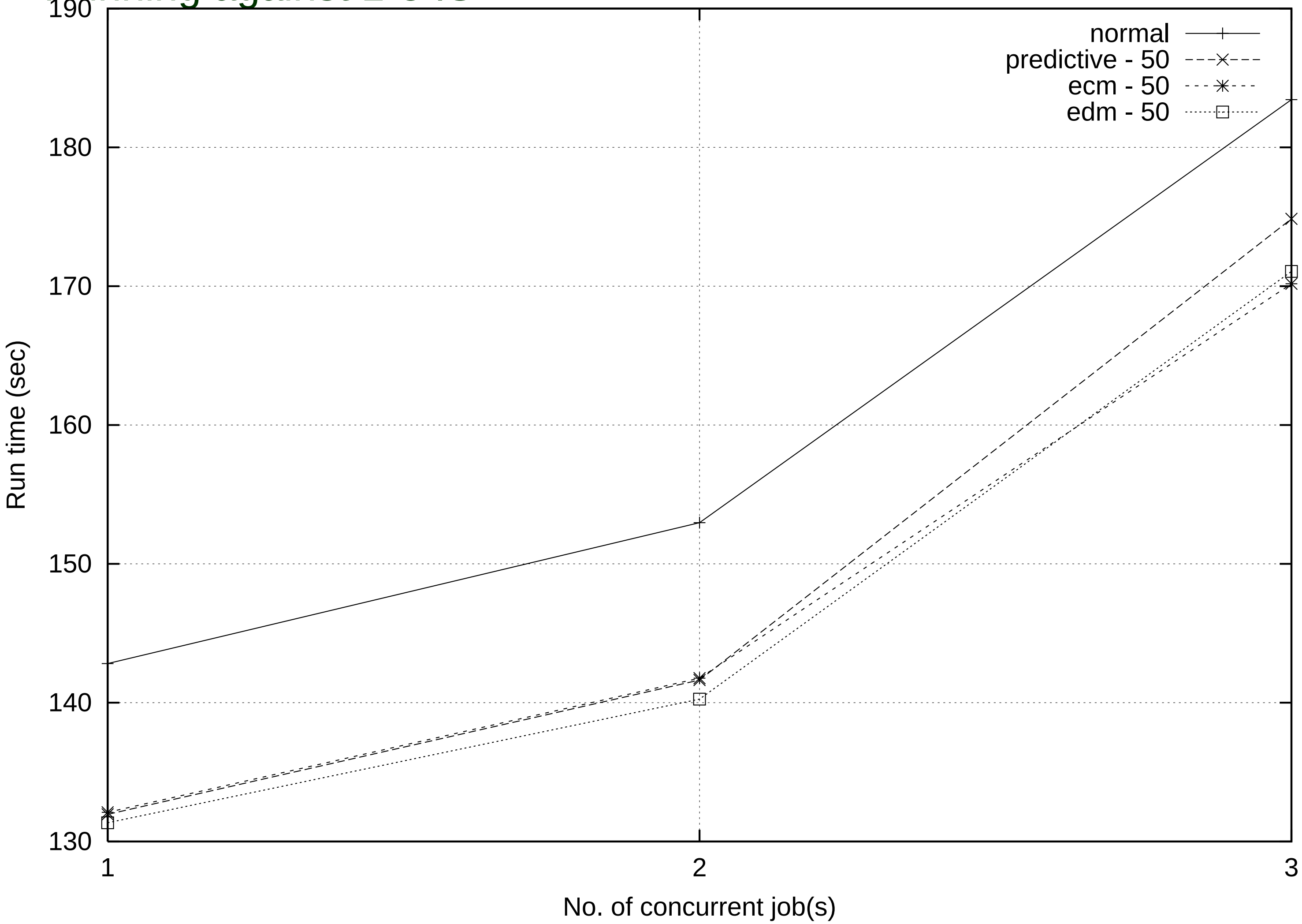
Impact of Coscheduling to Sequential task

- Running against 1-3 MG



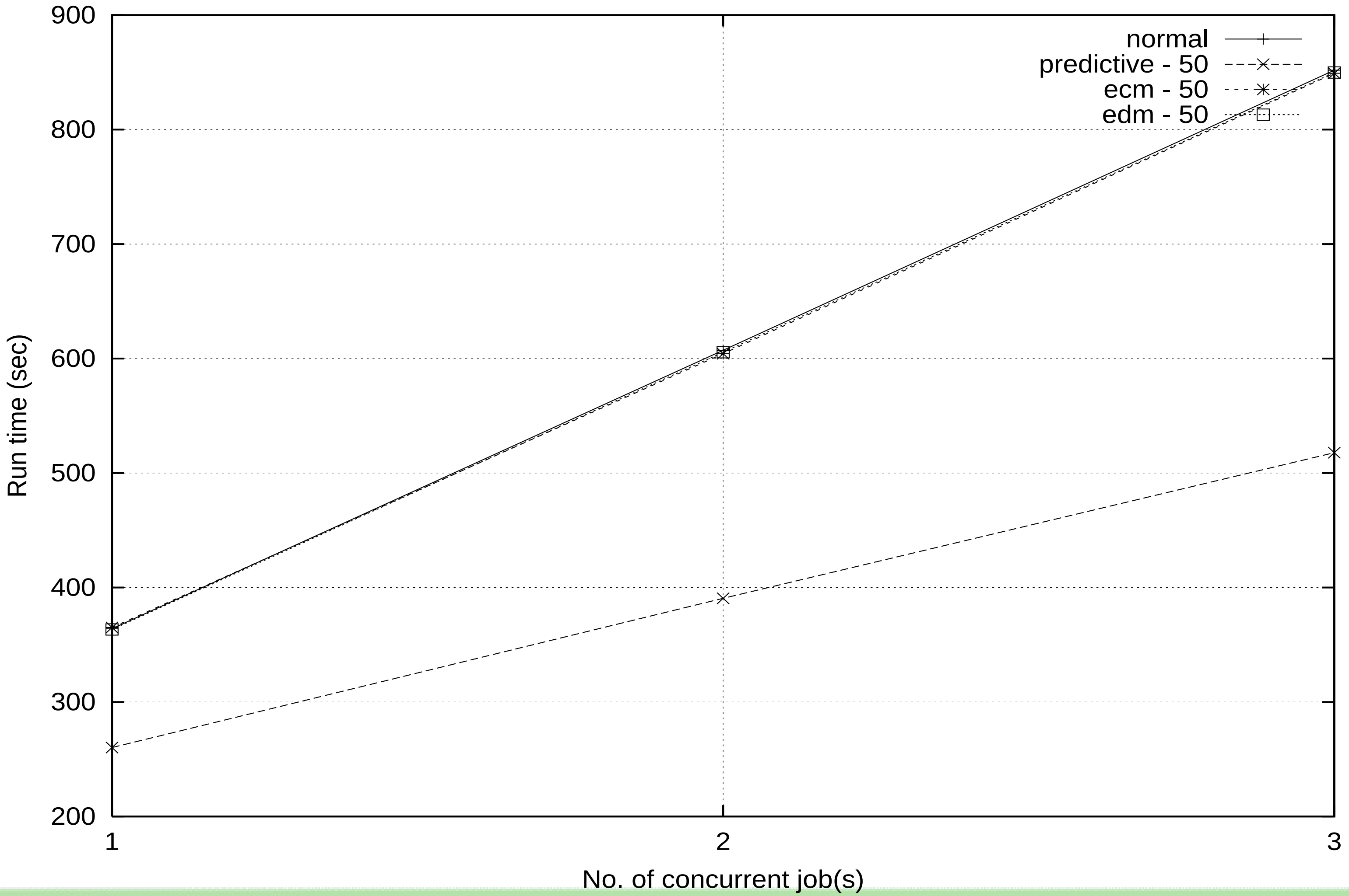
Impact of Coscheduling to Sequential task

- Running against 1-3 IS



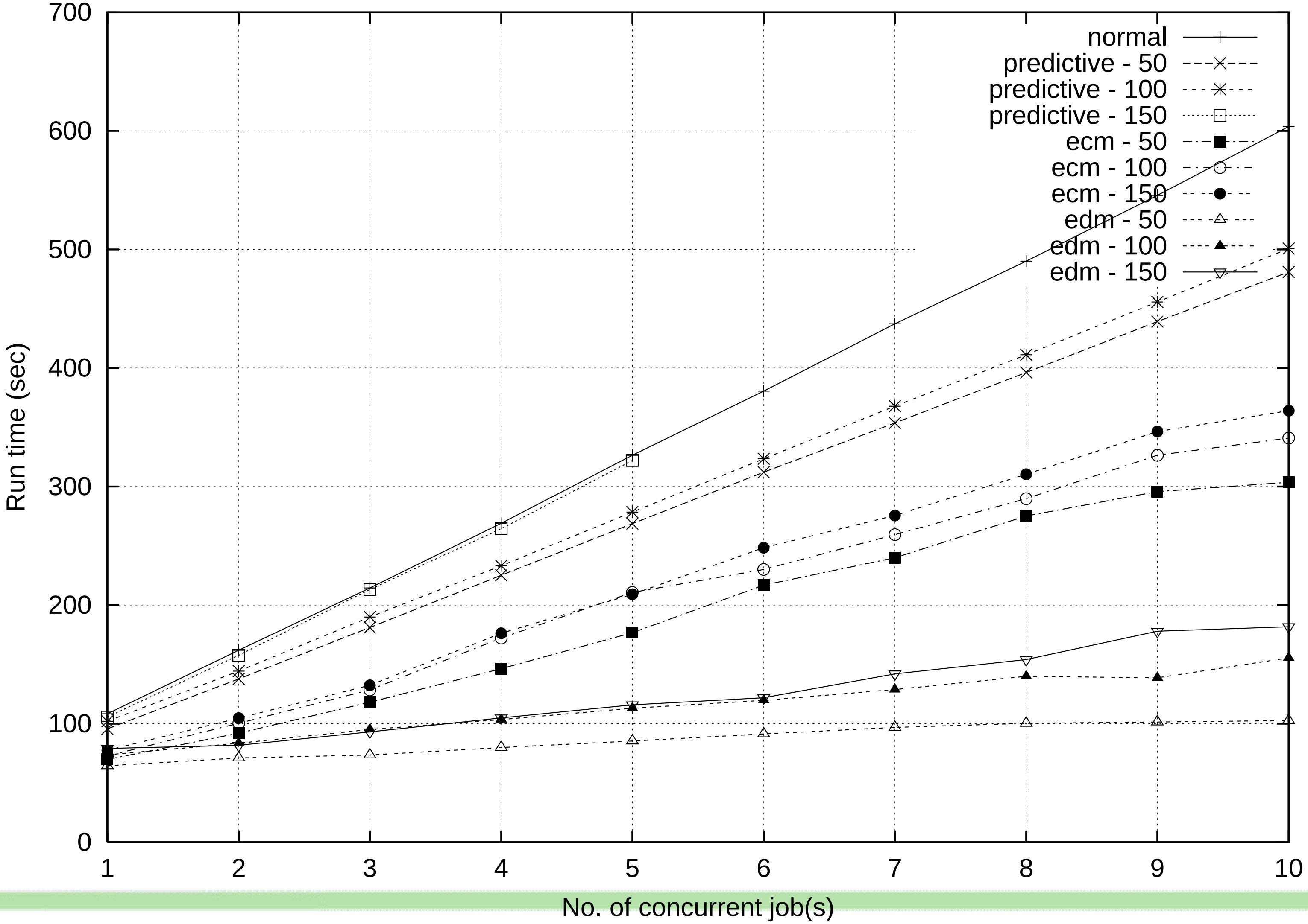
Impact of Coscheduling to Sequential task

- Running against 1-3 LU



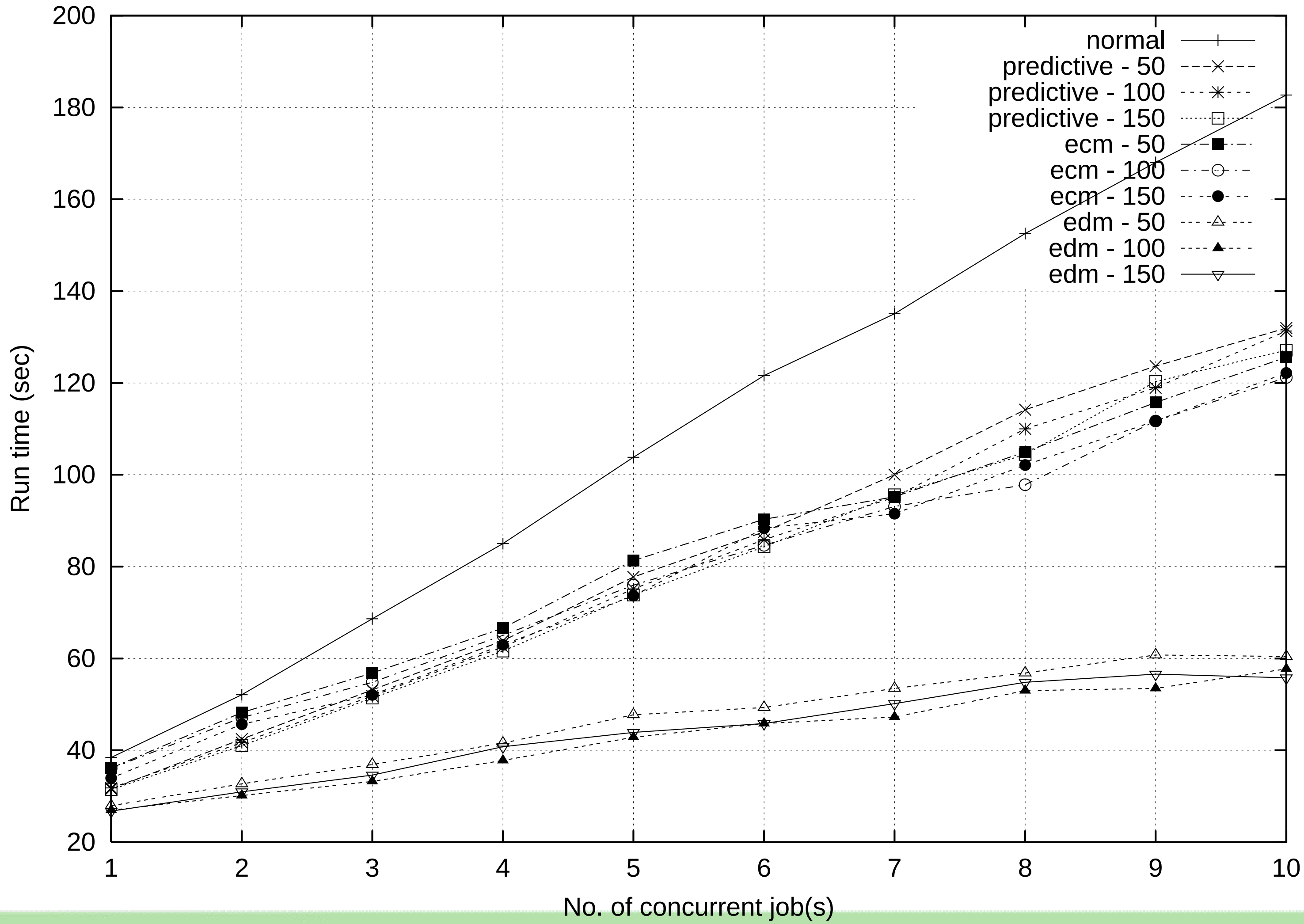
Scheduling Interval Variation

- Impact of different scheduling interval (MG with multiple sequential)



Scheduling Interval Variation

- Impact of different scheduling interval (MG with multiple sequential)



Discussion

- EDM perform the best when running against multiple sequential program, it performs better in case of fine-grain parallel program. However, no improvement occurs with coarse-grain
- ECM perform better than Predictive. It perform the best when running coarse-grain program. However, run-time of fine-grain program is worse when running against many fine-grain
- There is almost no overhead occurs in sequential program while running against coscheduled program. Because coscheduling just make benefits from the waste cycle.
- The choice of whether using ECM or EDM is depends on the system. ECM for coarse-grain and EDM for fine-grain
- Inspecting the process characteristics as the whole is better than using only snap-shot of statistics

Conclusion

- Our contribution
  - Propose the Energy Model for Implicit Coscheduling that can improve performance of parallel programs.
  - Develop a practical implementation in kernel to demonstrates that coscheduling is an applicable solution in Beowulf Cluster environment

Future Work

- Running the coscheduling framework in large-scale cluster and evaluate the performance behavior of such system.
- Use more process parameter to create a better algorithm. For example, I/O rate, trashing rate.
- Incorporate coscheduling with resource allocation will potentially gain more utilization and performance

Question?