Deadline Scheduling with Priority for Client-Server Systems on the Grid

Eddy Caron, Pushpinder Kaur Chouhan, Frédéric Desprez

8 November 2004
GRID Computing

Outline

Outline

1. Introduction
   - Grid Overview
   - Notations and constraints
2. Client-Server Scheduler
3. Simulation
4. Inference

Grid and GridRPC

- Grid : Platform resulted from aggregating distributed computers and storage units
  - Renting computation power and memory capacity
  - Need of Problem Solving Environments
- GridRPC : Implement Remote Procedure Call model over the Grid
  - Good and simple paradigm to implement the Grid
  - Run computation remotely

Grid and GridRPC

- Grid : Platform resulted from aggregating distributed computers and storage units
  - Renting computation power and memory capacity
  - Need of Problem Solving Environments
- GridRPC : Implement Remote Procedure Call model over the Grid
  - Good and simple paradigm to implement the Grid
  - Run computation remotely

Grid and GridRPC

- Grid : Platform resulted from aggregating distributed computers and storage units
  - Renting computation power and memory capacity
  - Need of Problem Solving Environments
- GridRPC : Implement Remote Procedure Call model over the Grid
  - Good and simple paradigm to implement the Grid
  - Run computation remotely

Grid and GridRPC

- Grid : Platform resulted from aggregating distributed computers and storage units
  - Renting computation power and memory capacity
  - Need of Problem Solving Environments
- GridRPC : Implement Remote Procedure Call model over the Grid
  - Good and simple paradigm to implement the Grid
  - Run computation remotely

Grid and GridRPC

- Grid : Platform resulted from aggregating distributed computers and storage units
  - Renting computation power and memory capacity
  - Need of Problem Solving Environments
- GridRPC : Implement Remote Procedure Call model over the Grid
  - Good and simple paradigm to implement the Grid
  - Run computation remotely

Grid and GridRPC

- Grid : Platform resulted from aggregating distributed computers and storage units
  - Renting computation power and memory capacity
  - Need of Problem Solving Environments
- GridRPC : Implement Remote Procedure Call model over the Grid
  - Good and simple paradigm to implement the Grid
  - Run computation remotely

Introduction

- Goal : To find a scheduling algorithm that can consider both priority and deadline of a task.
- Foundation : A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid by Takefusa et al.
- Motivation : Allocate the tasks on server where they can meet their deadline with in the priority constraint.
- Requirement : Performance prediction tool FAST and modelization

Introduction

- Goal : To find a scheduling algorithm that can consider both priority and deadline of a task.
- Foundation : A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid by Takefusa et al.
- Motivation : Allocate the tasks on server where they can meet their deadline with in the priority constraint.
- Requirement : Performance prediction tool FAST and modelization

Introduction

- Goal : To find a scheduling algorithm that can consider both priority and deadline of a task.
- Foundation : A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid by Takefusa et al.
- Motivation : Allocate the tasks on server where they can meet their deadline with in the priority constraint.
- Requirement : Performance prediction tool FAST and modelization

Introduction

- Goal : To find a scheduling algorithm that can consider both priority and deadline of a task.
- Foundation : A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid by Takefusa et al.
- Motivation : Allocate the tasks on server where they can meet their deadline with in the priority constraint.
- Requirement : Performance prediction tool FAST and modelization

Notation

$$T_{aS_i} = \frac{W_{send}}{P_{send}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{P_S}$$

- T_{aS_i}: the execution time for the task a on server i.
- W_{send}: the size of the data transmitted from the client to the server
- P_{send}: the predicted network throughputs from the client to the server
- P_{recv}: the predicted network throughputs from the server to the client and P_S is the server performance
- W_{recv}: the data size transmitted from the server to the client
- W_{aS_i}: the number of floating point operations of the task
- P_S: the server performance (floating point operations per second)

Notation

$$T_{aS_i} = \frac{W_{send}}{P_{send}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{P_S}$$

- T_{aS_i}: the execution time for the task a on server i.
- W_{send}: the size of the data transmitted from the client to the server
- P_{send}: the predicted network throughputs from the client to the server
- P_{recv}: the predicted network throughputs from the server to the client and P_S is the server performance
- W_{recv}: the data size transmitted from the server to the client
- W_{aS_i}: the number of floating point operations of the task
- P_S: the server performance (floating point operations per second)

Notation

$$T_{aS_i} = \frac{W_{send}}{P_{send}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{P_S}$$

- T_{aS_i}: the execution time for the task a on server i.
- W_{send}: the size of the data transmitted from the client to the server
- P_{send}: the predicted network throughputs from the client to the server
- P_{recv}: the predicted network throughputs from the server to the client and P_S is the server performance
- W_{recv}: the data size transmitted from the server to the client
- W_{aS_i}: the number of floating point operations of the task
- P_S: the server performance (floating point operations per second)

Notation

$$T_{aS_i} = \frac{W_{send}}{P_{send}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{P_S}$$

- T_{aS_i}: the execution time for the task a on server i.
- W_{send}: the size of the data transmitted from the client to the server
- P_{send}: the predicted network throughputs from the client to the server
- P_{recv}: the predicted network throughputs from the server to the client and P_S is the server performance
- W_{recv}: the data size transmitted from the server to the client
- W_{aS_i}: the number of floating point operations of the task
- P_S: the server performance (floating point operations per second)

Notation

$$T_{aS_i} = \frac{W_{send}}{P_{send}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{P_S}$$

- T_{aS_i}: the execution time for the task a on server i.
- W_{send}: the size of the data transmitted from the client to the server
- P_{send}: the predicted network throughputs from the client to the server
- P_{recv}: the predicted network throughputs from the server to the client and P_S is the server performance
- W_{recv}: the data size transmitted from the server to the client
- W_{aS_i}: the number of floating point operations of the task
- P_S: the server performance (floating point operations per second)

Notation

$$T_{aS_i} = \frac{W_{send}}{P_{send}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{P_S}$$

- T_{aS_i}: the execution time for the task a on server i.
- W_{send}: the size of the data transmitted from the client to the server
- P_{send}: the predicted network throughputs from the client to the server
- P_{recv}: the predicted network throughputs from the server to the client and P_S is the server performance
- W_{recv}: the data size transmitted from the server to the client
- W_{aS_i}: the number of floating point operations of the task
- P_S: the server performance (floating point operations per second)

Notation

$$T_{aS_i} = \frac{W_{send}}{P_{send}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{P_S}$$

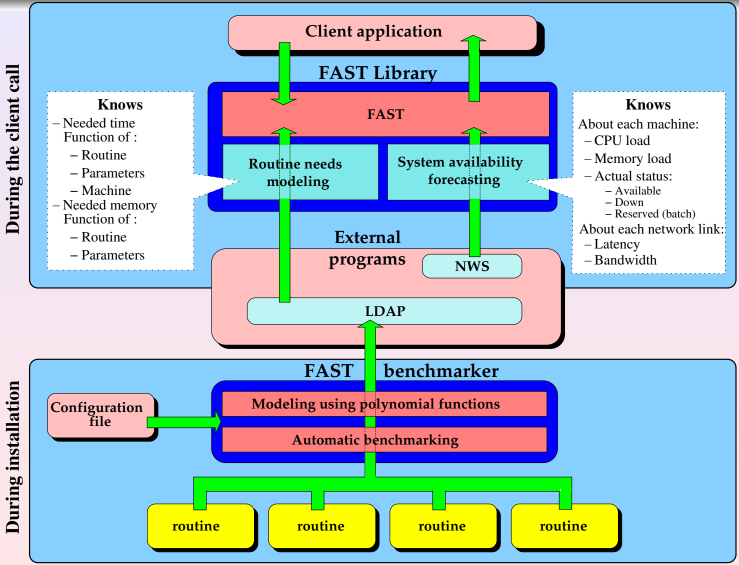
- T_{aS_i}: the execution time for the task a on server i.
- W_{send}: the size of the data transmitted from the client to the server
- P_{send}: the predicted network throughputs from the client to the server
- P_{recv}: the predicted network throughputs from the server to the client and P_S is the server performance
- W_{recv}: the data size transmitted from the server to the client
- W_{aS_i}: the number of floating point operations of the task
- P_S: the server performance (floating point operations per second)

Outline

Straightforward algorithm

repeat
  for all server S_i **do**
    if can_do(S_i, T_a) **then**
      $T_{aS_i} = \frac{W_{send}}{F_{Bd}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{F_{S_i}}$
      List=sort_insert(List,T_{aS_i}, T_a, S_i)
    end if
  end for
  num_submit=task_ack($List, \frac{2 \times W_{send}}{F_{Bd}}$)
  task_submit($List[num\_submit]$)
until the end

FAST's overview



Scheduling with Forecast Correction Mechanism

- $$T_{aS_i} = \frac{T_{aS_i} \times CorrecFAST}{100}$$

- $$CorrecFAST = \frac{nb\_exec \times CorrecFAST + \frac{100 \times T_r}{T_{aS_i}}}{nb\_exec + 1}$$

- $CorrecFAST$: an error average between the prediction time and the actual execution time.
- T_r : the actual execution time
- T_{aS_i}: the time predicted execution time

Scheduling with Forecast Correction Mechanism

- $$T_{aS_i} = \frac{T_{aS_i} \times CorrecFAST}{100}$$
- $$CorrecFAST = \frac{nb\_exec \times CorrecFAST + \frac{100 \times T_r}{T_{aS_i}}}{nb\_exec + 1}$$
- $CorrecFAST$: an error average between the prediction time and the actual execution time.
- T_r : the actual execution time
- T_{aS_i}: the time predicted execution time

Scheduling with Forecast Correction Mechanism

- $$T_{aS_i} = \frac{T_{aS_i} \times CorrecFAST}{100}$$
- $$CorrecFAST = \frac{nb\_exec \times CorrecFAST + \frac{100 \times T_r}{T_{aS_i}}}{nb\_exec + 1}$$
- $CorrecFAST$: an error average between the prediction time and the actual execution time.
- T_r : the actual execution time
- T_{aS_i}: the time predicted execution time

Scheduling with Forecast Correction Mechanism

- $$T_{aS_i} = \frac{T_{aS_i} \times CorrecFAST}{100}$$
- $$CorrecFAST = \frac{nb\_exec \times CorrecFAST + \frac{100 \times T_r}{T_{aS_i}}}{nb\_exec + 1}$$
- $CorrecFAST$: an error average between the prediction time and the actual execution time.
- T_r : the actual execution time
- T_{aS_i}: the time predicted execution time

Scheduling with Forecast Correction Mechanism

- $$T_{aS_i} = \frac{T_{aS_i} \times CorrecFAST}{100}$$
- $$CorrecFAST = \frac{nb\_exec \times CorrecFAST + \frac{100 \times T_r}{T_{aS_i}}}{nb\_exec + 1}$$
- $CorrecFAST$: an error average between the prediction time and the actual execution time.
- T_r : the actual execution time
- T_{aS_i}: the time predicted execution time

Scheduling with Forecast Correction Mechanism

$CorrecFAST = 100$
$nb\_exec = 0$
for all server S_i **do**
    if can_do(S_i, T_a) **then**
        $T_{aS_i} = \frac{W_{send}}{F_{Bd}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{F_{S_i}}$
        $T_{aS_i} = \frac{T_{aS_i} \times CorrecFAST}{100}$
        $List$=sort_insert$(List, T_{aS_i}, T_a, S_i)$
    end if
end for
num_submit=task_ack$(List, \frac{2 \times W_{send}}{F_{Bd}})$
T_r=task_submit$(List[num_submit])$
$CorrecFAST = \frac{nb\_exec \times CorrecFAST + \frac{100 \times T_r}{T_{aS_i}}}{nb\_exec + 1}$
$nb\_exec{+}{+}$

Priority and Deadline Mechanism

```
repeat
    for all server S_i do
        if can_do(S_i, T_a) then
            T_{aS_i} = W_send / F_Bd + W_recv / P_recv + W_{aS_i} / F_{S_i}
        end if
        if T_{aS_i} < TD_a then
            count_fallback_tasks(T_a, T_{aS_i}, TP_a, TD_a)
            if TF_{aS_i} < TD_a then
                best_server(S_i,best_server_name)
            end if
        end if
    end for
    task_submit(best_server_name,task_name)
    Re-submission(task_name)
until the end
```

Priority and Deadline Mechanism

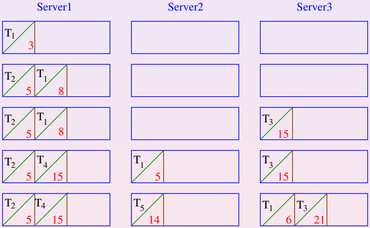
- can_do This function returns `true` if server S_i have the resource required to compute task T_a. This function takes into account the availability of memory and disk storage, the computational library etc.
- Count_fallback_tasks This function counts the fallbacked tasks. Tasks that cannot meet their deadline after the insertion of the new task are called fallbacked tasks. Task T_a is placed according to its priority TP_a on the server task queue, which may change the execution time of the tasks on the queue.

Priority and Deadline Mechanism

- best_server This function selects the best server among the servers that can execute the task within the deadline time. The best server is selected by comparing the number of fallbacked tasks. Server with less fallbacked tasks is selected for the task execution. If the servers have same number of fallbacked tasks, then the time to compute the task is compared and the server that takes less time is selected.
- task_submit It performs the remote execution on the server given by the `best_server` and the argument of the function is one server and not a list of servers.
- Re-submission This function submits the fallbacked task to the servers, for recomputing the execution time. If any server can meet the task's deadline then the task is allocated to that server.

Priority and Deadline Mechanism

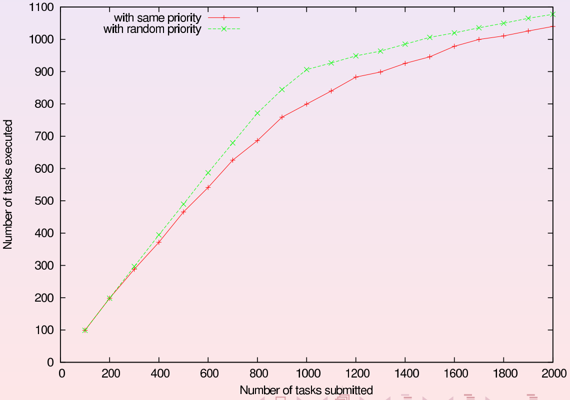
			Exec. time on server		
Task	Priority	Deadline	S_1	S_2	S_3
1	3	15	3	5	6
2	5	10	5	12	9
3	2	30	11	20	15
4	4	20	10	np	17
5	5	15	12	14	np



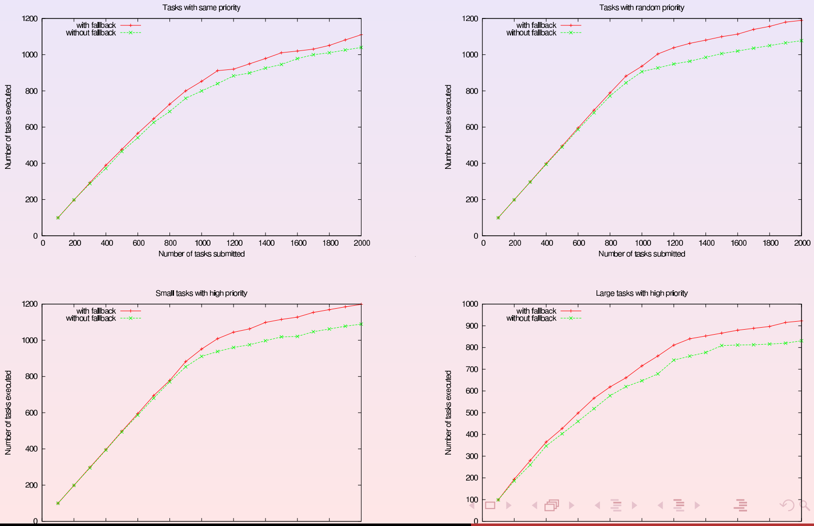
Outline

Priority based tasks are executed without fallback mechanism

Testbed 100 servers, priority range 1-10, deadline = 5 $\times T_{aS_i}$



Tasks executed with and without fallback



Outline

Inference

- Conclusion
  - Introduction of the performance prediction tool FAST
    - http://graal.ens-lyon.fr/FAST
  - Focused on the management of tasks with respect to their priorities and deadlines
  - Increase the number of tasks that can meet their deadlines by
    - Fallback mechanism to reschedule tasks
    - Load correction mechanism (using FAST)
    - Using task's priority
- Future Work
  - Implement these sheduling algorithms in DIET platform
    - ASP middleware
    - Grid-RPC middleware
    - http://graal.ens-lyon.fr/DIET