Efficient Feasibility Analysis of DAG Scheduling with Timing Constraints in the Presence of Faults

Author:

Xiaotong Cui, Jun Zhang, Kaijie Wu, Edwin Sha.

College of Computer Science, Chongqing Univ., Chongqing, China.

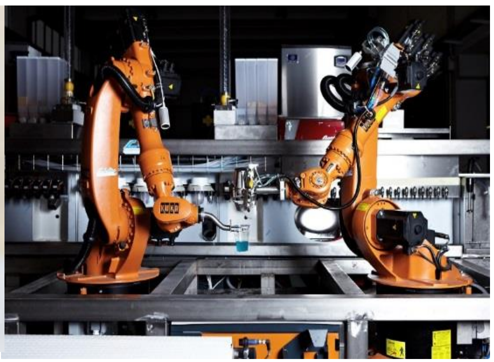
Outline

- **Introduction**
  - Real-time systems and applications
  - DAG scheduling
  - Fault-tolerance
- **Problem Definition**
- **Our Technique**
- **Experiments**
- **Conclusion**

Real-Time systems and Applications

- Real-time systems and applications become more common in our lives.

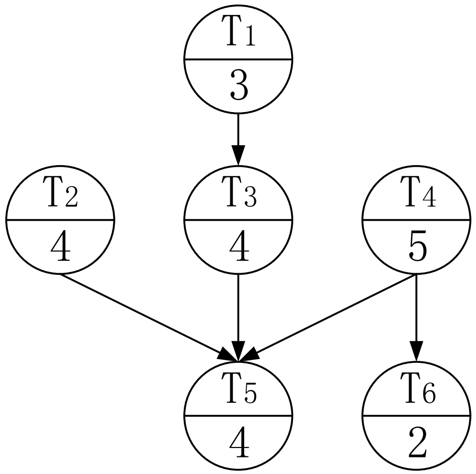





- The total correctness of an operation in real-time systems depends upon:
  - its logical correctness
  - the time it used

DAG Scheduling on Multi-Cores

- We usually use weighted **directed acyclic graph (WDAG)** to model a set of tasks.



data dependency

Communication delay on
The same processor: 0
Different processors: 1

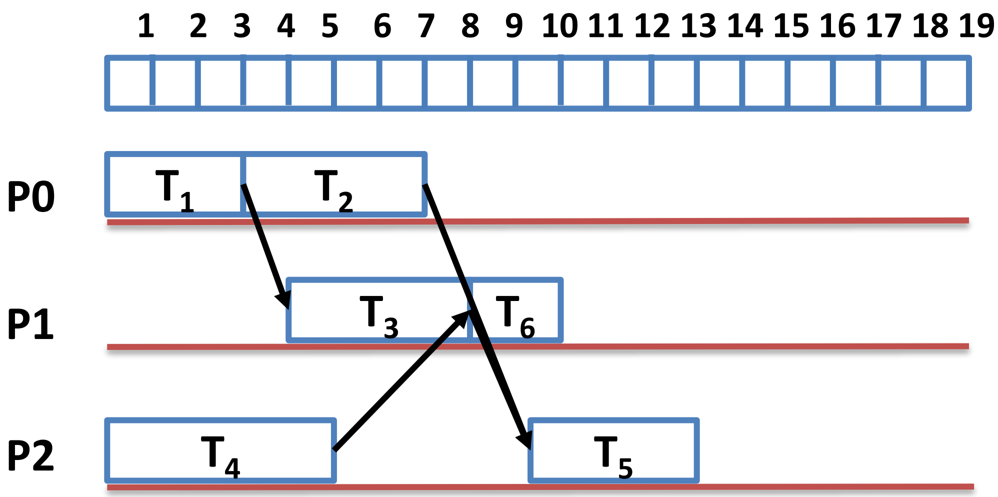


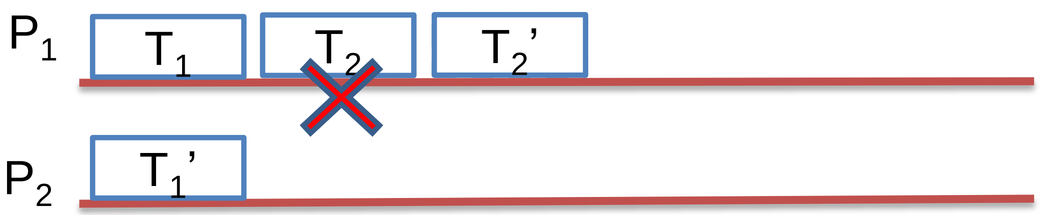
Fig. A scheduled task set

Guaranteeing logical correctness

- However, the execution of a task may not be successful sometimes.
- In real-time systems, finishing execution before deadline is important though, logical correctness is fundamental.
- So it is necessary to provide **fault-tolerance** in real-time schedules.

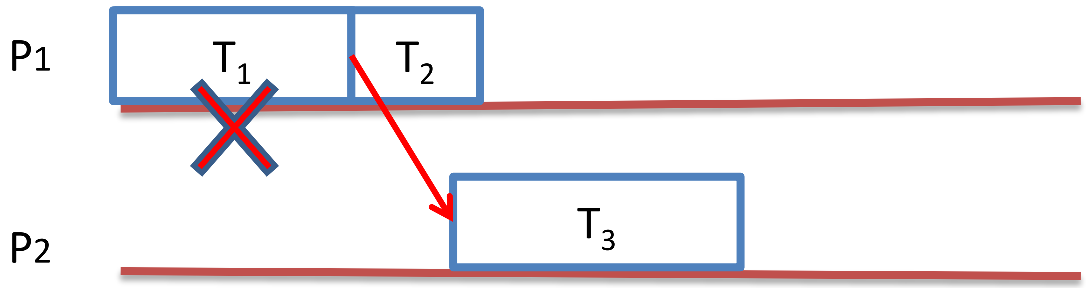
Fault-tolerant Mechanism

- Fault-tolerance is built into the schedule using two types of backups:
  - Active backup
  - Passive backup



Fault-tolerant Mechanism

- For the sake of simplicity and energy conservation, in our problem, we use passive backup to build **fault-tolerance following two rules**.
  - Backup is executed immediately after the error task;
  - All tasks dependent on it are delayed.



■ It should be noted that the re-executed task may incur error again.

Problem Definition

- In multi-core systems, when given
  - A scheduled task set with N related tasks, say T={t1, t2, …, tn} ;
  - the maximum number of faults that could occur during the execution frame in the system, say X.

- **Then, what's the worst-case finish time(WCFT) of this scheduled task set ?**
- After finding WCFT, then the feasibility can be found, too.

Problem complexity

- Exactly, if a task set consists of N tasks and is subject to a maximum number of X faults, there would be

$$\binom{N+X-1}{X}$$

distinct cases of fault occurrences.

- With a given fault occurrence, to compute the WCFT of the scheduled task set, the time complexity will be $O(N^2)$, which is the longest path of the scheduled task set.

- So, the total time complexity would be

$$O\left(\frac{(N+X-1)!}{X!\,(N-1)!}N^2\right)$$

if all cases are computed which is very exhaustive.

Properties of a task set

- For a task set modeled by WDAG, we add two dummy vertices.
  - Tsr : has an edge for each vertex that has no incoming edge in original DAG,
  - Tsk: has an edge for each vertex that has no outgoing edge in original DAG.
  - One task T's critical paths: the longest paths from Tsr to T
- Let Tc be the current task under investigation,

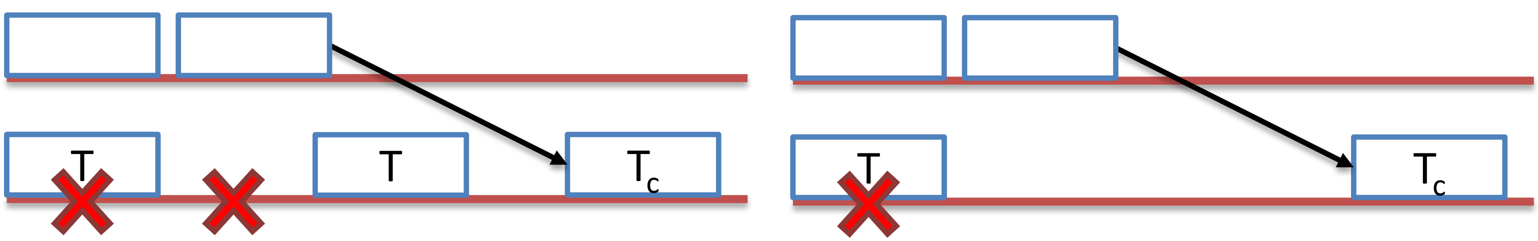
  PS_{Tc} be the Parents Set of Tc,

  AS_{Tc} be the Ancestors Set of Tc.

  Obviously, $PS_{Tc} \subseteq AS_{Tc}$,

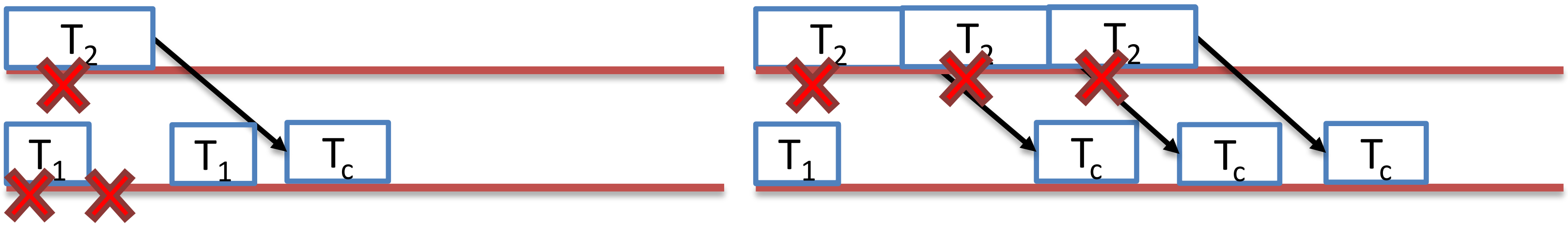
Some Critical Proofs

- Lemma 1: T is not in any critical paths of Tc, if critical paths of Tc don't change when T incurs x faults, reducing x will not affect the finish time of Tc.



Some Critical Proofs

- Lemma 2: if more than one task in ASTc incur faults, a worse or status quo finish time of Tc can be always be found by letting one task T∈ASTc incur all the X faults.



Theorem: there exists at least one critical task for any task Tc such that if this task incurs all the expected X faults, task Tc experiences its worst-case finish time.

Preprocessing

- When given a scheduled task set and its original data dependency modeled by DAG, there are two kinds of dependency.
  - data dependency
  - schedule dependency

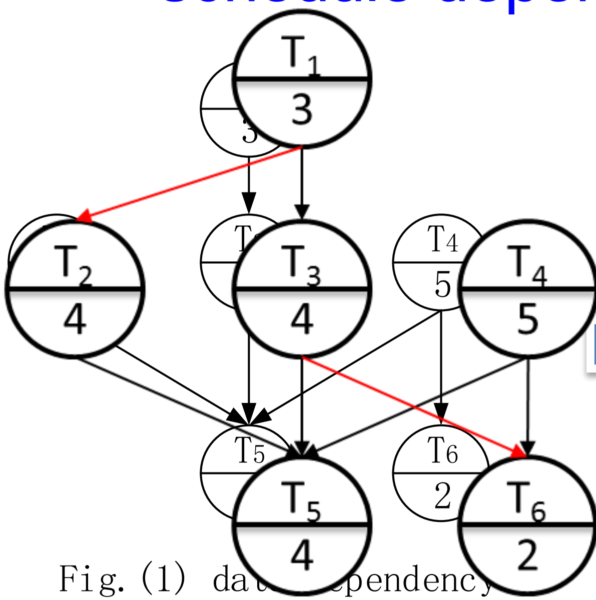


Fig. (1) data dependency

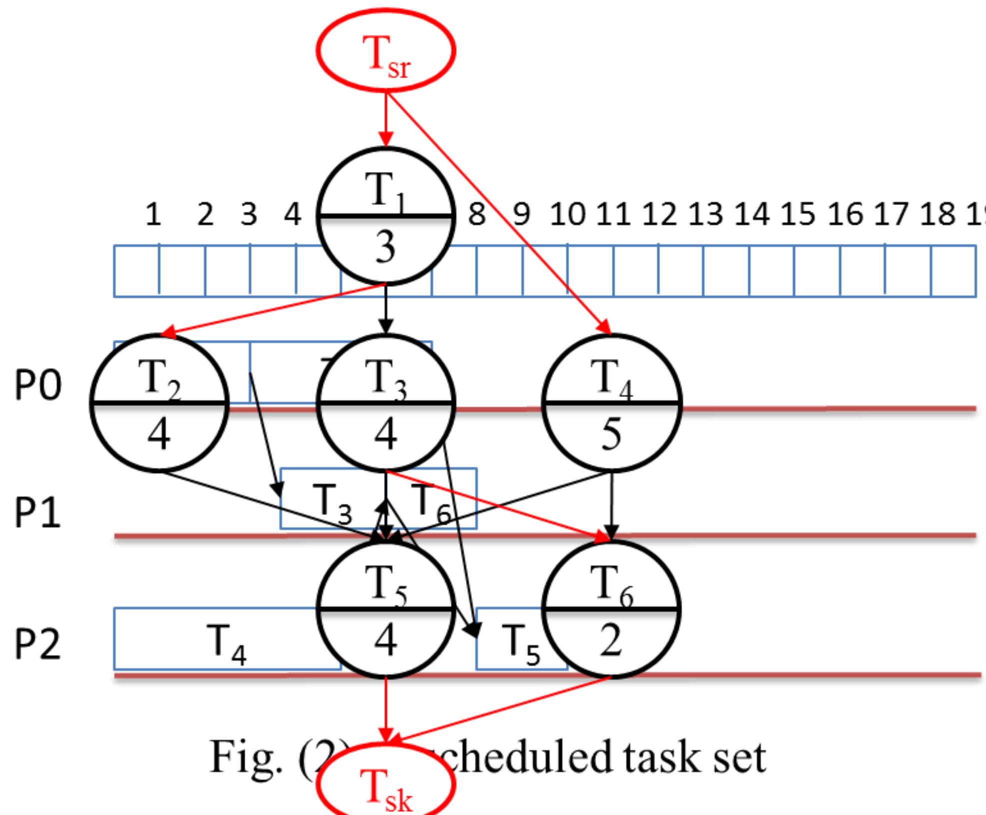
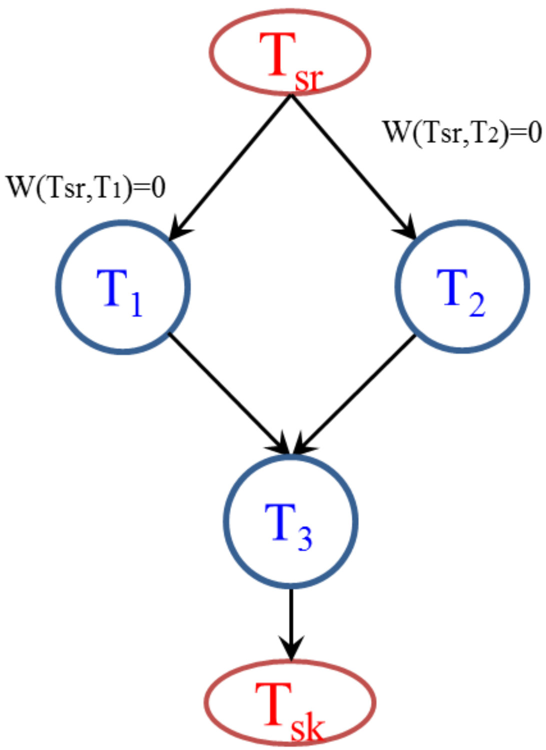


Fig. (2) scheduled task set

Our Technique

- Base on our theorem, we can solve the problem recursively.

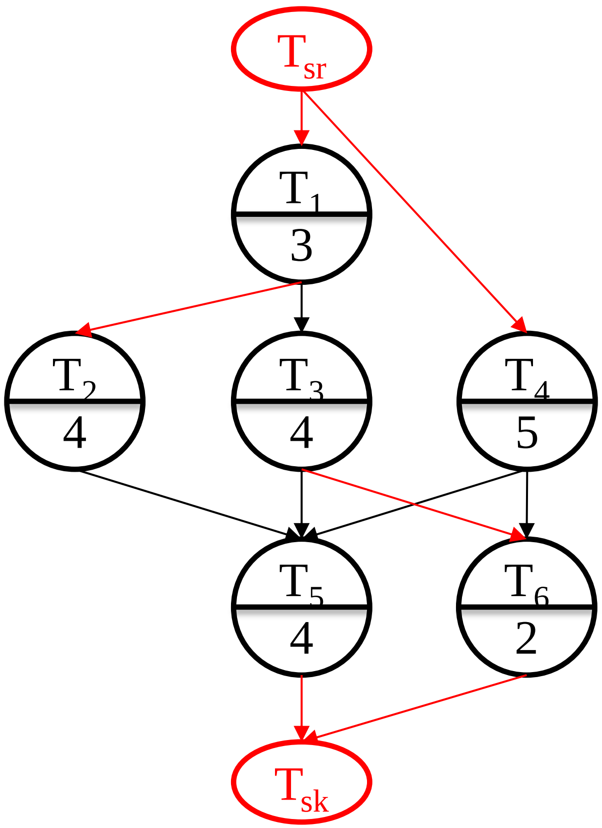


$WCFT(T_1)=C_{T1}+X*C_{T1}$ $\quad$ $CT(T_1)=T_1$

$WCFT(T_2)=C_{T2}+X*C_{T2}$ $\quad$ $CT(T_2)=T_2$

$$WCFT(T_3)=\mathit{Max}\begin{cases} WCFT(T_1)+W(T_1,T_2)+C_{T3} \\ WCFT(T_2)+W(T_1,T_2)+C_{T3} \\ BCFT(T_3)+X*C_{T3} \end{cases}$$

Recursively solve the problem



Ancestors set of task T_{sk}

$$WCFT(T_{sk}) = max \begin{cases} WCFT(T_5) + W(T_5, T) + C_{Tsk} \\ WCFT(T_6) + W(T_5, T) + C_{Tsk} \\ BCFT(T_{sk}) + X * C_{Tsk} \end{cases}$$

$IF \quad WCFT(T_{sk})= WCFT(T_5) + W(T_5, T) + C_{Tsk}$

CT(T_{sk})=CT(T_5)

Else

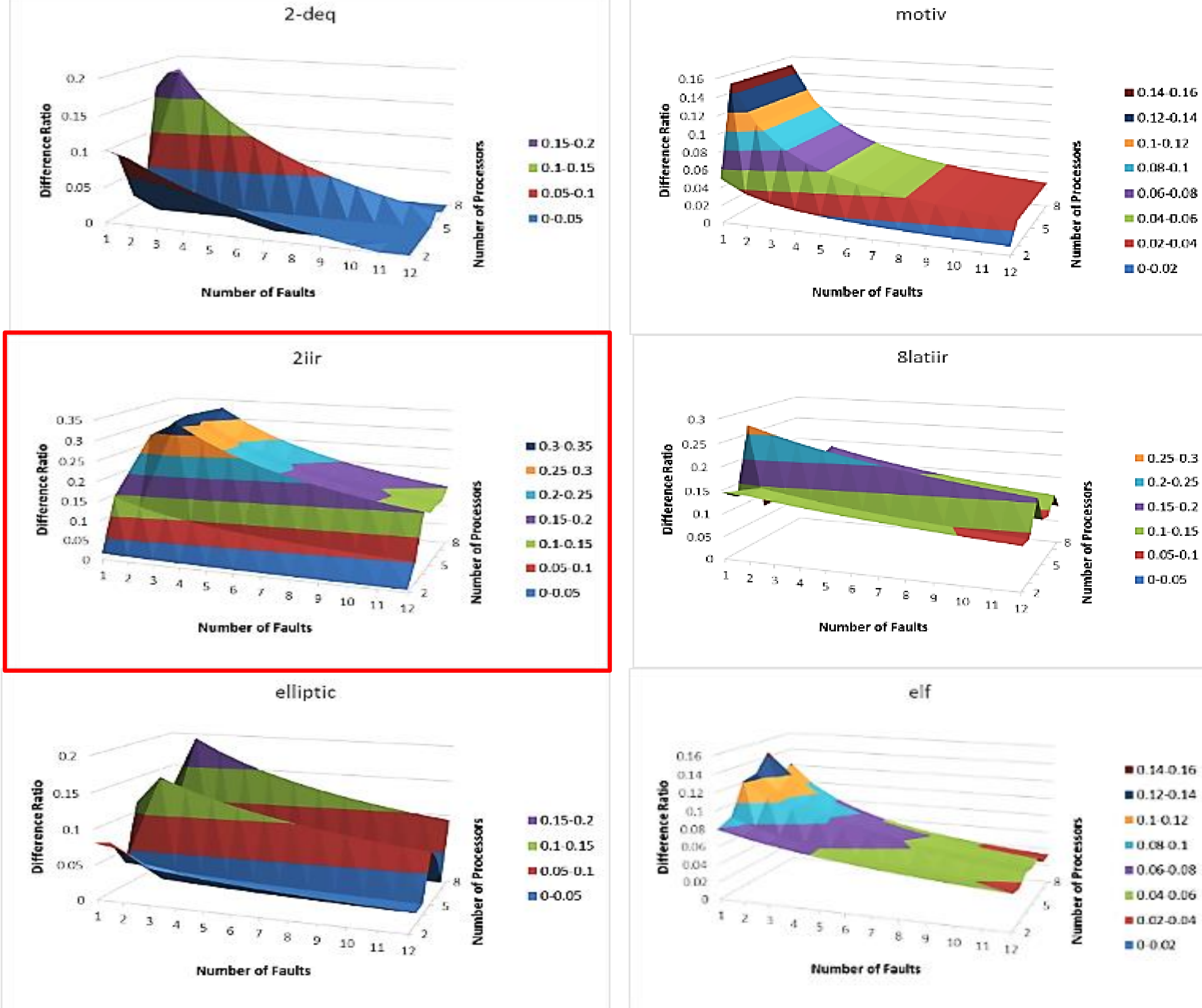
CT(T_{sk})=CT(T_6)

Solve the problem recursively

task T_{sk}

Experimental Setup

- A common practice that bet WCFT using the task with longest execution time could under-estimate the finish time of the task set.
- We compare the WCFT obtained by our algorithm with the finish time when all faults occur on the task with the longest execution time.
- Six benchmarks from DSPstone are used.



2-deq
Difference Ratio
Number of Faults
Number of Processors
0.15-0.2
0.1-0.15
0.05-0.1
0-0.05
motiv
Difference Ratio
Number of Faults
Number of Processors
0.14-0.16
0.12-0.14
0.1-0.12
0.08-0.1
0.06-0.08
0.04-0.06
0.02-0.04
0-0.02
2iir
Difference Ratio
Number of Faults
Number of Processors
0.3-0.35
0.25-0.3
0.2-0.25
0.15-0.2
0.1-0.15
0.05-0.1
0-0.05
8latiir
Difference Ratio
Number of Faults
Number of Processors
0.25-0.3
0.2-0.25
0.15-0.2
0.1-0.15
0.05-0.1
0-0.05
elliptic
Difference Ratio
Number of Faults
Number of Processors
0.15-0.2
0.1-0.15
0.05-0.1
0-0.05
elf
Difference Ratio
Number of Faults
Number of Processors
0.14-0.16
0.12-0.14
0.1-0.12
0.08-0.1
0.06-0.08
0.04-0.06
0.02-0.04
0-0.02

Conclusion

- Given a task set with N tasks and X being the maximum number of faults could occur, we conclude that there exists at least one critical task for each task.
- A task undergoes it worst-case finish time when one of its critical tasks incurs all X faults.
- We propose a recursive algorithm which can identify the critical task and the worst-case finish time of a scheduled task set.
- For a task set with N tasks and The algorithm takes only $O(N^2)$.

Thank You!