

Temporal Verification in Grid Workflow Systems

Jinjun Chen
 Swinburne University of Technology, Australia
<http://www.ict.swin.edu.au/personal/jchen>
 October 2006



Content



- Introduction
- Multiple temporal consistency states
- Assigning fine-grained temporal constraints
- Selecting sufficient yet necessary checkpoints
- Temporal dependency based temporal verification
- Exception handling
- Future work



Introduction



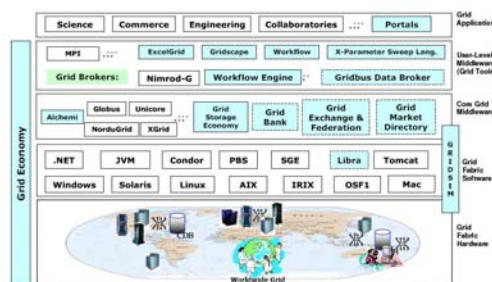
- In grid architecture, a grid workflow system is a type of user-level middleware which aims to support large-scale complex scientific and business processes in many e-science and e-business applications such as climate modelling, disaster recovery or high energy physics.



Introduction



- From www.gridbus.org



Introduction



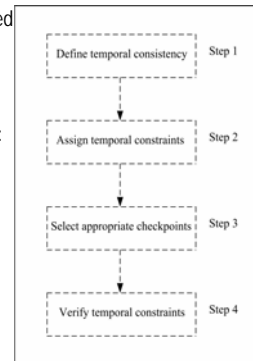
- In reality, complex scientific and business processes are normally time constrained. Hence, time constraints are often set when they are modelled as grid workflow specifications.
- Temporal constraints mainly include: upper bound, lower bound and fixed-time
 - An upper bound constraint between two activities is a relative value that the duration between them is less than or equal to.
 - A lower bound constraint between two activities is a relative value that the duration between them is greater than or equal to.
 - A fixed-time constraint at an activity is an absolute value by which the activity must be completed.



Introduction



- Temporal verification is used to identified any temporal violations so that we can handle them in time.
- Temporal verification cycle:



Introduction



- Step 1 is to define temporal consistency. At this step, temporal consistency states are defined .
- Step 2 is to assign temporal constraints. Users may first set some coarse-grained temporal constraints. Based on them, we may then assign some fine-grained ones during specific grid workflow specification and execution.
- Step 3 is to select appropriate checkpoints for conducting temporal verification. A checkpoint is an activity point where we need to verify temporal constraints.



Introduction



- Step 4 is to verify temporal constraints.
- Note: We take upper bound constraints to conduct corresponding discussion.*
 - A lower bound constraint is symmetrical to an upper bound constraint.
 - A fixed-time constraint is a special case of upper bound constraints whose start activity is just the first activity of the whole grid workflow.



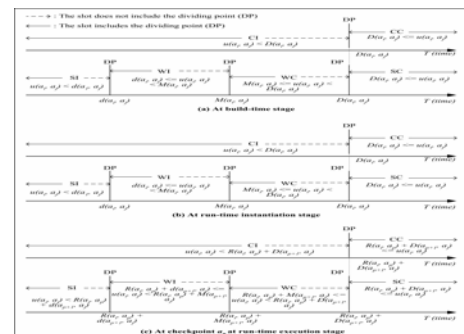
Multiple Temporal Consistency States



- Conventional verification work defines two consistency: consistency and inconsistency, denoted as CC (Conventional Consistency) and CI (Conventional Inconsistency).
- We argue that we should divide them into 4 temporal consistency states: Strong Consistency (SC), Weak Consistency (WC), Weak Inconsistency (WI) and Strong Inconsistency (SI).
- CC corresponds to SC while CI is divided into WC, WI and SI.



Multiple Temporal Consistency States



Multiple Temporal Consistency States



- SC corresponds to CC
- WC can be adjusted without triggering any exception handling
- WI can be handled by simpler exception handling
- SI has to be handled by complex and costly exception handling as that in the conventional work for handling CI
- Conventional work handles SC, WC, WI and SI in the same way, i.e. by the complex and costly exception handling.



Assigning fine-grained temporal constraints



- Some processes only have one end-to-end upper bound constraint.
- This would be intuitive and simple from user perspective.
- Not sufficient to control grid workflow execution in terms of time.
 - Lasts a long time
 - Highly dynamic



Selecting sufficient and necessary checkpoints



- Verifying temporal constraints at every activity is not efficient as we may not have to do so at some activities such as those which can be completed within allowed time intervals.
- Checkpoint Selection Strategies (CSS).
- 7 representative checkpoint selection strategies have been developed. But, they often ignore some necessary checkpoints and/or select some unnecessary ones.



Selecting sufficient and necessary checkpoints



- Ignored checkpoints will result in some temporal verification omitted. Unnecessary checkpoints will cause some unnecessary temporal verification.
- Clearly, neither is desirable. So, we may ask: Can we develop a strategy which only selects sufficient and necessary checkpoints?
- We answer the question positively by presenting such one.



Selecting sufficient and necessary checkpoints



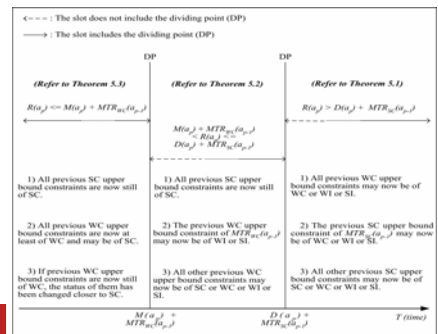
- Minimum Time Redundancy: The minimum one of all time redundancies of all upper bound constraints.
 - SC time redundancy
 - WC time redundancy.
- Relationship between SC & WC time redundancy and SC, WC, WI & SI.



Selecting sufficient and necessary checkpoints



- Relationships...



Selecting sufficient and necessary checkpoints



- Three conclusions from the above figure
- Conclusion 1 for $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$: We need to verify all previous SC and WC upper bound constraints. There is at least one previous SC upper bound constraint which is violated and now is not of SC. It is exactly the one whose SC time redundancy at a_{p-1} is $MTR_{SC}(a_{p-1})$.



Selecting sufficient and necessary checkpoints



- Conclusion 2 for $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1})$: We need not verify all previous SC upper bound constraints, only all previous WC ones. And there is at least one previous WC upper bound constraint which is violated and now is not of SC and WC. It is exactly the one whose WC time redundancy at a_{p-1} is $MTR_{WC}(a_{p-1})$.



Selecting sufficient and necessary checkpoints



- Conclusion 3 for $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$: We need not verify all previous SC upper bound constraints. As to previous WC ones, their status has been changed closer to SC (can even be changed to SC sometimes). Therefore, if a previous WC upper bound constraint is still of WC after execution of a_p , the previous handling or adjustment on it can be carried forward. Hence, we need not do anything further to it. That is to say, we need not verify it.



Selecting sufficient and necessary checkpoints



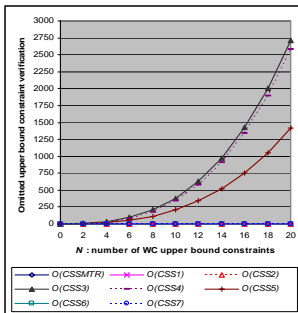
- Our strategy: CSS_{MTR}
- At activity a_p :
 - If $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, we take it as a checkpoint for verifying SC, WC, WI & SI of all previous SC upper bound constraints, and for verifying WC, WI & SI of all previous WC upper bound constraints;
 - If $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1})$, we take it as a checkpoint for verifying SC, WC, WI & SI of all previous WC upper bound constraints only;
 - If $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$, we do not take a_p as a checkpoint.



Selecting sufficient and necessary checkpoints



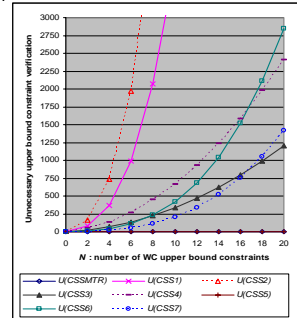
- Quantitative Evaluation
- Omitted verification



Selecting sufficient and necessary checkpoints



- Unnecessary verification



Selecting sufficient and necessary checkpoints



- With our strategy, we only select sufficient and necessary checkpoints. The quantitative evaluation has shown that our strategy can improve overall temporal verification effectiveness and efficiency significantly.



Temporal dependency based temporal verification



- Multiple temporal constraints are dependent on each other in terms of their verification.
- This is because later verification may make previous verification ineffective and also later verification may utilise previous verification results to save current verification computation for better efficiency.
- We need to investigate temporal dependency between temporal constraints and its profound impact on overall temporal verification effectiveness and efficiency.



Temporal dependency based temporal verification



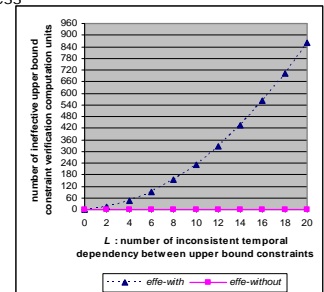
- At build-time and run-time instantiation stages, we need to verify temporal constraints as well as their mutual temporal dependency. For better verification effectiveness.
- At run-time execution stage, we deploy temporal dependency so that we can utilise previous verification to save current verification computation for better verification efficiency.



Temporal dependency based temporal verification



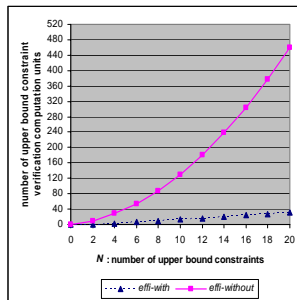
- Evaluation
- Verification effectiveness



Temporal dependency based temporal verification



- Verification efficiency



Temporal dependency based temporal verification



- With temporal dependency, we can improve overall temporal verification effectiveness and efficiency significantly.



Exception handling



- Handling temporal verification results, i.e. SC, WC, WI and SI.
 - For SC, nothing needs to be done.
 - For WC, we utilise potential time redundancy of later activity execution to adjust it so that it can still be kept as SC.
 - For WI, we can rely on simpler exception handling (future work)
 - For SI, we can deploy the complex and costly exception handling in conventional work.



Future work



- Exception handling for WI and SI
- Integrate theoretical results into some existing grid workflow systems
- Apply theoretical results into real-world applications
- New time models for representing timed grid workflow



Questions



- Thanks for your patience and attention

- Questions ?

