

Middleware-Enhanced Concurrency of Transactions

Interactive Responsiveness and Concurrent Workflow

Transactional Cascade® Technology Paper

Ivan Klianov, Managing Director & CTO



Published in November 2005 | Updated in March 2010

Copyright © 2005-2010 Transactum
All rights reserved

Abstract

Application scaling is not linear and deteriorates the responsiveness. Enhancing the concurrency of serialized database transactions mitigates the non-linearity. Implementation on a workflow system with *open nested transaction* model of execution and *transaction chopping* shortens lock intervals and increase concurrency of transactions. As a result, scaling curves become steeper. Steeper scaling curve affects positively process application responsiveness in few aspects. It improves simultaneously both application throughput and responsiveness, everything else being equal. Furthermore, it enables application scaling to a new *Marginal Efficiency* barrier with better or at least the same responsiveness. Transactional Cascade® workflow technology provides means necessary to make scaling curve steeper. Moreover, with Transactional Cascade® way of scaling and a steeper curve, scaling is not limited by a new *Marginal Efficiency* barrier. It goes further up to a new *Marginal Latency* barrier, where higher volume of concurrent users could be served with better or acceptable responsiveness.

Table of Contents

Abstract	2
Table of Contents	2
Executive Summary	3
1. Transactions Length and Application Performance	4
1.1. Long Transactions Enforce Sequential Performance	4
1.2. Short Transactions Promote Parallel Performance	5
2. Workflow and Concurrent Transactions	6
2.1. Transaction Chopping and Concurrency	6
2.2. Transaction Chopping and Workflow	6
2.3. Open Nested Transactions and Data Consistency	7
3. Concurrent Transactions and Responsiveness	8
3.1. Concurrent Transactions and Application Throughput	8
3.2. Application Throughput and Process Latency	10
3.2. Process Capacity, Marginal Costs, and Marginal Latency	11
4. Conclusion	12

Executive Summary

Maintaining the responsiveness with scaling is a notoriously elusive task. Scaling a process with serialized database operations adds further complexity with the necessity of both consistency of data and concurrency of transactions. This paper explores the approach of giving up the complete transaction isolation to enhance the concurrency of serialized transactions. The goal is to maintain process responsiveness during more extended scaling.

Updating multiple data aggregations in one serialized transaction causes the most severe performance bottleneck. Its only excuse is the guaranteed data consistency. In context of Internet, however, this approach is not acceptable. Creation of Internet-scale business applications require part of the responsibility for consistency of process data to be shifted from transaction managers to workflow systems.

Concurrency of execution of multiple instances of a transaction with multiple serialized operations might be enhanced with implementation of *transaction chopping* on a workflow system with *open nested transaction* model of execution. This is what Transactional Cascade® workflow technology does. It shortens lock intervals to the extreme and supports a transaction execution model that diminishes the potential for inconsistency.

With Transactional Cascade® technology, as a result of improved concurrency of transactions, the process-scaling curve becomes steeper. Steeper scaling curve affects positively process responsiveness in few aspects. It improves simultaneously both application throughput and responsiveness, everything else being equal. Application can interact with more concurrent users. At the same time, it responds each individual user interaction with shorter time interval.

The steeper curve provides space for additional scaling before hitting a new *Marginal Efficiency* barrier. Throughout the additional scaling interval, responsiveness is better than it was at the point of old *Marginal Efficiency* barrier. With Transactional Cascade® technology there is no efficiency barrier. Scaling is meaningful up to the *Marginal Latency* barrier – the last scaling point with acceptable responsiveness, which is reached at a considerably later stage.

Giving up the complete transaction isolation is a necessary condition for concurrency of transactions and Internet-scaling. Better transactional concurrency simultaneously increases application throughput and responsiveness. Transactional Cascade® technology provides means for enhanced concurrency of serialized transactions without compromising the consistency of data.

1. Transactions Length and Application Performance

1.1. Long Transactions Enforce Sequential Performance

A process that performs its database operations in one or more long transactions in effect imposes a sequentially executing element on the process application

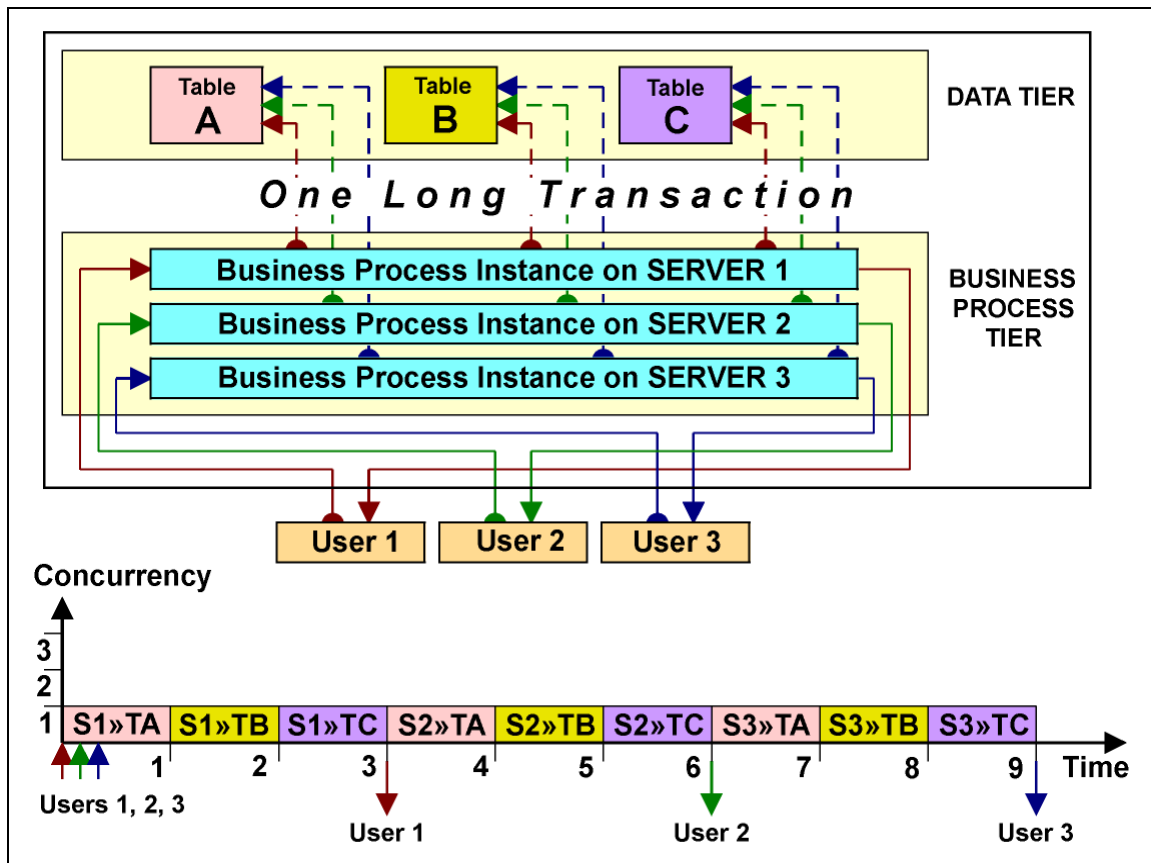


Figure 1 – Sequential execution of instances of a business process with a long database transaction

Figure 1 presents 3 instances of a business process that performs database writing in 3 tables within one long transaction, where writing in each table requires serialization. When a process instance on Server 1 acquires a lock on Table A and starts the transaction, process instances on Server 2 and Server 3 will halt waiting for release of the lock on Table A.

Lock on Table A will be released after process instance on Server 1 completes its writing in Table A, Table B, and Table C and the transaction is committed. The same will happen when process instance on Server 2 acquires the lock on Table A, and then process instance on Server 3 acquires it, etc. Clearly, at any point in time only one process instance will perform and the other two process instances will wait.

1.2. Short Transactions Promote Parallel Performance

A process that performs its database operations in multiple short transactions in effect enables a degree of transactional parallelism, as each individual short transaction might be executed concurrently with the other short transactions.

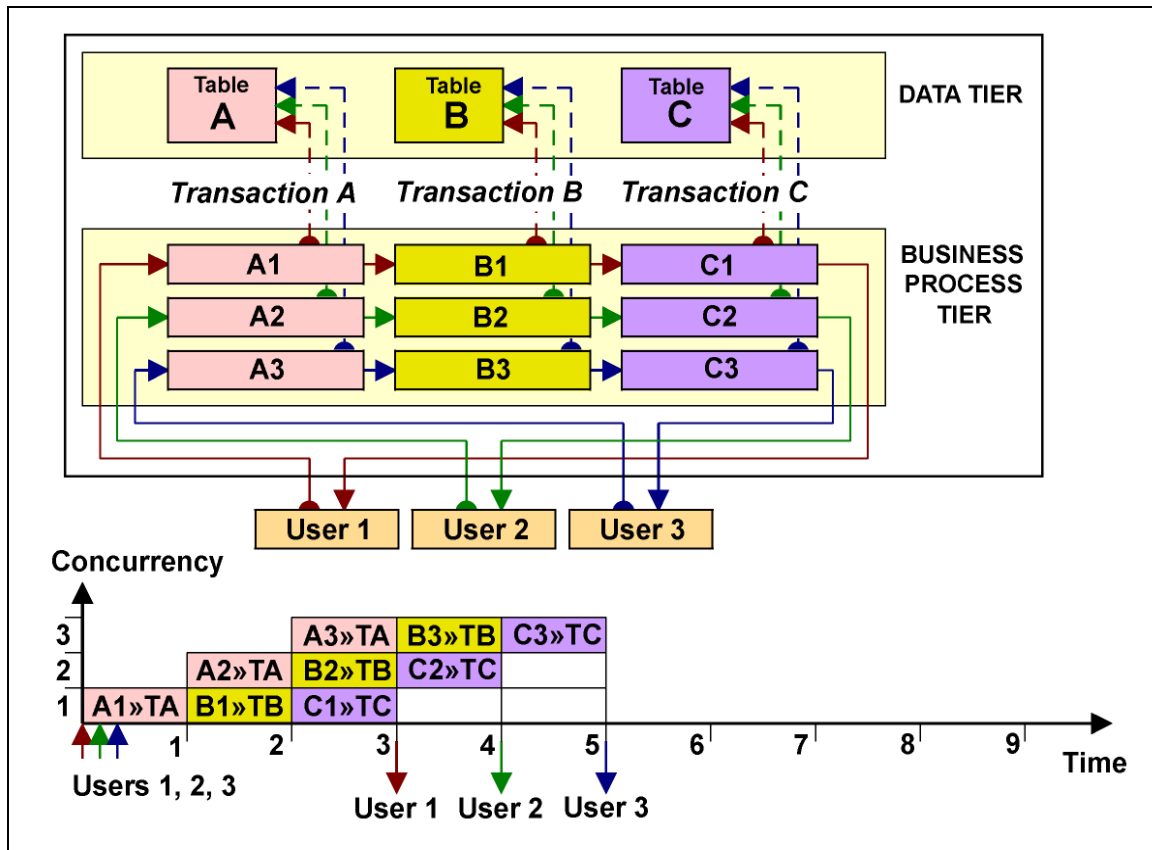


Figure 2 – Parallel execution of instances of a business process with 3 short database transactions

Figure 1 presents 3 instances of a business process that performs database writing in 3 tables with 3 short transactions, where writing in each table requires serialization. When a process instance on Server 1 acquires a lock on Table A and starts the transaction, process instances on Server 2 and Server 3 will halt waiting for release of the lock on Table A.

Lock on Table A will be released after process instance on Server 1 completes its writing in Table A and transaction is committed. At the same time when the process instance on Server 1 acquires a lock on Table B and starts a transaction, the process instance on Server 2 acquires the released lock on Table A and starts a transaction.

On next step, when both transactions commit, the process instance on Server 1 acquires a lock on Table C and starts a transaction, the process instance on Server 2 acquires a lock on Table B and starts a transaction, and the process instance on Server 3 acquires a lock on Table A and starts a transaction. Thereby, all three transactions are performed concurrently, or in parallel if each table is created on a separate disk drive.

2. Workflow and Concurrent Transactions

2.1. Transaction Chopping and Concurrency

In 1995, *Shasha et al*¹ proposed a simple algorithm to partition transactions into the smallest possible pieces with the following property: If the small pieces of transaction instances execute serialized, then the transaction instances will appear to execute serialized. The goal is to shorten lock intervals without sacrificing serialized execution, or in other words, to guarantee serialized execution without paying for it with long intervals of locking.

Chopping algorithm permits more concurrent operations with database systems. It is useful where locks contention exists. Alternatively, if system resources are heavily loaded or locking conflicts do not exist, there's no reason for chopping. Doing it will only add to transaction starting and commit overheads with no throughput improvement.

2.2. Transaction Chopping and Workflow

With regard to shortened lock intervals, the algorithm of transaction chopping is best implemented with the **open nested transaction**² model. With it, the results of actions in a sub-tree of transactions become available immediately after individual steps are committed, rather than waiting for the entire transaction to complete. Thereby, locks are acquired only for the duration of individual transactions rather than for the duration of controlling transaction.

Transactional Cascade® workflow technology implements the open nested transaction model with chopped database transactions. Workflow system chain layer performs the function of higher-level transaction. Chain layer is responsible for both atomicity and for durability of higher-level transactions in progress. It is implemented with message queuing transactions.

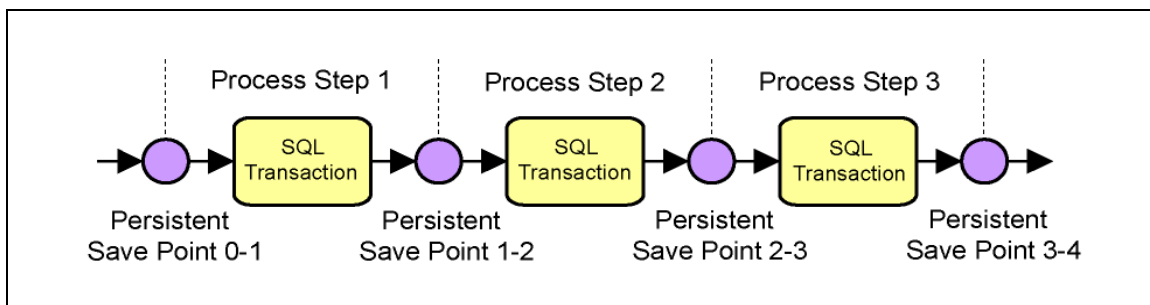


Figure 3 – Chained Execution of SQL Transactions

On figure 3, each SQL transaction is nested in a message queuing (MQ) transaction for chaining. Chain layer performs in the following manner: A committed MQ transaction passes the processing context to a persisted save point, thereby implicitly starting next MQ transaction. With activated option for high-availability, the MQ transaction passes the processing context to a pair of persisted save point – on the local computer and on its backup buddy. In case of technical failure on the local computer, the workflow execution will continue on the backup buddy computer from the last successfully committed MQ transaction.

¹ Dennis Shasha, Francois Lirbat, Eric Simon, Patrick Valduriez. 1995. [Transaction Chopping: Algorithms and Performances Studies](#) *ACM Transactions on Database Systems*, 1995.

² Jim Gray, Andreas Reuter. 1993. *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann

2.3. Open Nested Transactions and Data Consistency³

A potential problem with open nested transactions is the possibility some transactions to see inconsistent data. This is the price for performance gains over close nested and long flat transactions. Eventual data inconsistency is usually due to the following sequence of events: (1) Transaction commits; (2) Data from committed transaction is read by another transaction; (3) The committed transaction is rolled back with execution of compensator.

Transaction managers do not guarantee the consistency of data, just assist it. Transactional Cascade® approach deals with consistency in a similar way. It specifies the conditions where seeing inconsistent data and acting upon it might in worst case enforce integrity constraints. If the operations of all SQL sub-transactions are commutative and associative, the implemented transaction will commit in a consistent state, but with undesirable semantic result.

For example, the highest cost for one instance of a transaction implemented with Transactional Cascade® seeing another instance's committed intermediate result before being compensated, is the possibility a sub-transaction of the first instance to enforce integrity constraints. As a result, the sub-transaction will commit with semantic failure and the first instance will commit in a consistent state, but with undesirable semantic result.

With Transactional Cascade® workflow, a semantic failure does not prevent transaction committing – just triggers branching of execution. Transactions abort for internal system reasons only. Aborted transactions are considered retry-able until committed. Few unsuccessful retries are regarded as failure that requires system restart.

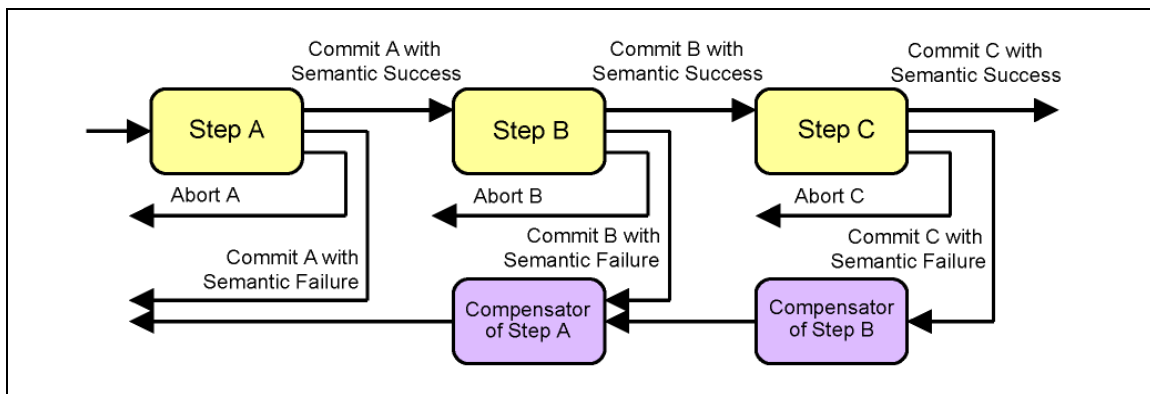


Figure 4 – Execution of Compensating Transactions

Figure 4 presents rolling back with execution of compensators. Transactional Cascade® requires a sub-transaction to persist its result AND the data necessary for undoing itself. For example, a committed sub-transaction that updates an aggregation involves execution of a compensator to undo the updating and requires availability of data about how exactly to undo.

Key Point

Where contention for serialization locks exists, transaction chopping and execution from a workflow system increases concurrency of database operations.

³ Suggested authority on how to handle data inconsistency: Shel Finkelstein, Dean Jacobs, Rainer Brendle. [Principles for Inconsistency](#) 4th Biennial Conference on Innovative Data Systems Research (CIDR), Jan. 2009, Asilomar, California, USA

3. Concurrent Transactions and Responsiveness

Application responsiveness is ability to perform the work necessary to process users' requests and prepare responses within a predefined interval of time.

Higher application throughput (which is a consequence of its processing capacity) means the ability to perform more work per unit of time. With a single application user, it does not deliver a tangible quality. With multiple users, however, it guarantees that the application has the capacity to maintain its responsiveness even when the volume of concurrent users becomes significantly higher.

When the number of concurrent application users is higher than the application throughput, the only way an application can cope with a workload in excess of its processing capacity is to queue the excessive requests and wait until free capacity is available. This in effect makes some requests to be processed with much longer delays, thereby strongly deteriorating the overall application responsiveness.

To prevent this, application might be scaled by connecting more infrastructure to increase its processing capacity. Initially, this step will solve the problem. Repeating these actions further, however, will gradually degrade the responsiveness to unacceptable level. Individual users will wait longer before receiving response as process latency increases faster than processing capacity and reaches a point where further scaling will make it higher than the critical level.

To avoid the first scenario, an application needs scaling of its processing capacity to ensure parallel processing of concurrent users. At the same time, to avoid the second scenario, process-scaling curve must become steeper; otherwise latency experienced by individual users will soon become higher than the critical value because of the marginal latency barrier.

3.1. Concurrent Transactions and Application Throughput

Application throughput is limited by the speed of its sequential items. Performance speed of a system with parallel and sequentially executed elements is limited by the speed of its sequential fraction ([Amdahl's law](#)) and some database transactions are an essential part of business applications that cannot be made fully parallel. For example, database transactions that involve updating of an aggregate value must be performed sequentially.

Therefore, to increase the processing capacity of an application, execution of additional process instances must be paired with additional parallelism of database transactions. In context of Internet scaling, the parallelism achievable with the standard method (of partitioning and physical disk per partition) is a fraction of needed. Number of process instances is generally a multiple of the number of realizable parallel transactions.

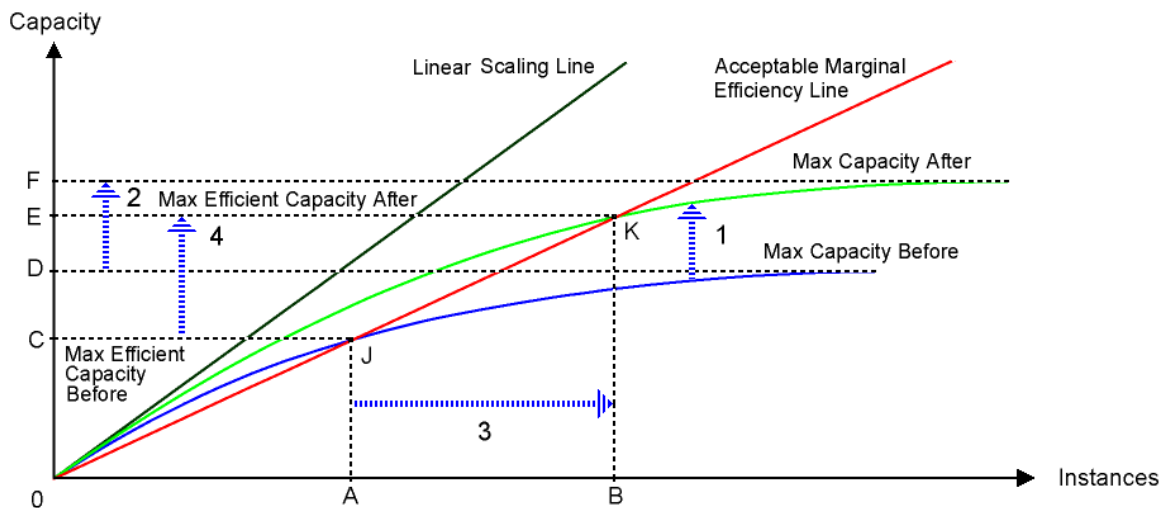


Figure 5 – Effects of Increased Transactional Parallelism on Scaling

Figure 6 presents a business process with non-linear scaling⁴ and the effects of increased parallelism of database transactions. The effect 1: higher parallelism pushes the non-linearity curve up. Effect 2 is a consequence of 1: the Max Capacity of scaling gets higher. Max Capacity of scaling has a practical value only when scaling with zero marginal efficiency costs. For example, when scaling a business process built with Transactional Cascade® technology.

Scaling with hardware parallelism is restricted by the increasing marginal costs – an additional unit of hardware runs an additional unit of process instances but results in creation of less than a unit of additional processing capacity. The Max Efficient Capacity is result of scaling of process instances up to the point of Acceptable Marginal Efficiency, which is a projection of the point of intersection between the scaling curve and the Acceptable Marginal Efficiency line.

Effect 3 is a consequence of 1: the new point of efficiency of scaling K shifts the Marginal Costs Barrier of this process from being at point A to being at point A. This allows more middleware infrastructure to be connected and more business process instances to be created and run. Effect 4 is a consequence of 3: execution of more business process instances increases the available processing capacity.

Increased parallelism of database transactions shifts the Marginal Costs Barrier and enables scaling of application infrastructure with acceptable marginal costs, thereby scaling the number of application instances and the maximum available processing capacity. The higher processing capacity permits servicing more application users without deterioration of responsiveness.

The Marginal Costs Barrier exists with middleware capable only of CPU-based parallelism. Effect 1 just increases Transactional Cascade® maximum processing capacity from the maximum of the original non-linearity curve to the maximum of the new non-linearity curve. Ability to scale an application to the absolute maximum with zero marginal costs of scaling, however, does not guarantee it is a practically meaningful exercise.

⁴ More details on non-linearity of application scaling are presented in Transactional Cascade® Concept Paper Three: [Scaling with Zero Marginal Costs](#).

3.2. Application Throughput and Process Latency

The interval that represents the average time necessary for application to process users' requests and prepare responses is the process latency. Normally, a process should be able to perform with a value of time interval being smaller than the value of its Maximum Acceptable Latency.

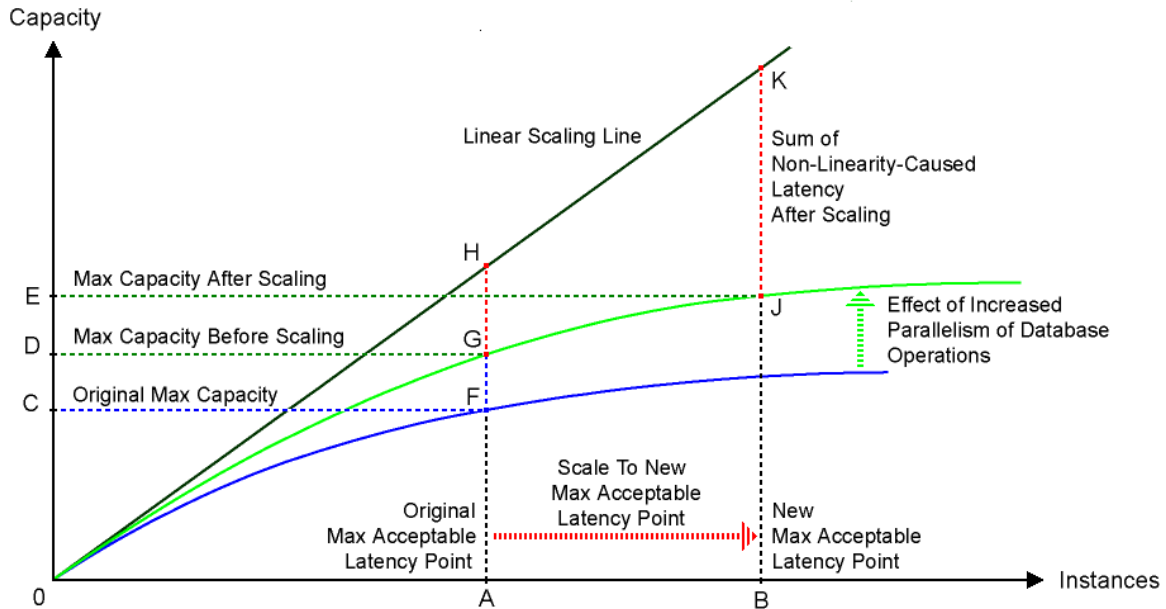


Figure 6 – Effects of Increased Transactional Parallelism on Latency

A latency of a business process has a fixed and a variable component. The variable latency is caused by the same reasons that cause the non-linearity of scaling. Consequently, sum of variable latencies of a number of users is proportional to the non-linearity-wasted part of the infrastructure necessary to process requests of these users.

For example, on figure 7 the process with scaling curve OF has maximum acceptable latency with number of process instances presented with point A. With this number of process instances, the available processing capacity is presented with point C and proportion of the infrastructure that is wasted because of the scaling non-linearity is presented with the interval HF. (The angle between Linear Scaling Line and horizontal axes is 45°.)

Curve OGJ is the scaling curve of the same process after the effect of increased parallelism of database transactions. The higher steepness of the new scaling curve has four major consequences:

1. The maximum processing capacity with the same number of instances (point A) increase from point C to point D;
2. Even though the application can now process more users concurrently, the sum of their variable latency is smaller – presented with the interval HG.
3. Number of process instances could be scaled to point B, where the individual user latency will still be smaller than the maximum acceptable latency.
4. After scaling, the maximum processing capacity with acceptable latency will increase from point D to point E.

3.2. Process Capacity, Marginal Costs and Marginal Latency

On figure 7, at point G the process has higher capacity (ability to process more concurrent users) and considerable smaller user-experienced latency compared with point F. At point J, the process has user-experienced latency equal to that of point F but with considerably more concurrent users. At any point on the curve segment GJ left of point J, the process will have even higher processing capacity and smaller user-experienced latency.

Point B on figure 7 represents Marginal Latency barrier, while point B on figure 6 represents Marginal Costs barrier. Business process scaling with middleware capable only of CPU-based parallelism is expensive. Consequently, the usual reason for applications' poor responsiveness is the insufficient scaling because of the Marginal Costs Barrier rather than scaling beyond the Marginal Latency barrier.

Business process scaling with a workflow-capable middleware, which can scale with no marginal costs like the Transactional Cascade® workflow technology does, can ensure:

- Maximum transactional parallelism.
- Higher steepness of the scaling curve.
- Right-shifted Marginal Latency barrier
- Scaling up to the new Marginal Latency barrier.

Key Point

For maximum responsiveness, a business process application needs workflow architecture for decomposition of transactions and ability to scale with no Marginal Costs barrier.

4. Conclusion

Responsiveness of business process applications could be improved by enhancing the concurrency of database transactions. With applications that experience contention for serialization locks, chopping a long serialized database transaction into a number of short transactions with chained execution increases the number of concurrently executed transactions. This method achieves shorter response time to individual requests and higher capacity – for processing of more concurrent users – even without infrastructure scaling.

Transactional Cascade® workflow technology permit realization of *transaction chopping* algorithms. It cuts down the transaction lock intervals to the extreme by implementation of *open nested transaction* model. At the same time, it assumes that part of the responsibility for consistency of process data is shifted from transaction managers to workflow systems. Chained execution of short transactions and separation of processing of aborted transactions from semantic failure of a business activity enables application architects to neutralize the potential for data inconsistency.

Transactional Cascade® workflow technology can execute high volume of concurrent business process instances per CPU core and, therefore, scales with zero marginal costs – up to the point with process latency equal to maximum acceptable. Enhancing the concurrency of database transactions provides a space for additional process scaling before process latency becomes equal to the maximum acceptable.

Value Proposition

Transactional Cascade® technology enables maximization of parallelism of database transactions and, thereby, additional process scaling until process latency becomes equal to the maximum acceptable, with no need for additional hardware and software licenses.

[Give a feedback or ask questions](#)

TRANSACTUM PTY LTD

Tel: + 61 2 9567 1150

Fax: + 61 2 9567 1160

WWW.TRANSACTUM.COM

PO Box 324, Brighton-Le-Sands NSW 2216, Australia