Lean and Efficient Business Tier for Performance Scaling

Muralidaran Natarajan
BIT Mesra
Mumbai, India

Nandlal L. Sarda
IIT Bombay
Bombay, India

Sharad C. Srivastava
BIT Mesra
Ranchi, India

ABSTRACT

Large mission critical real-time legacy OLTP systems supporting the service sectors like banking, telecom, and financial services are monolithic in nature and thereby not flexible to enable business transformation which is the need of the hour due to emerging dynamic changes to business ecosystem. Broadly speaking, these large mission critical applications can be classified into three stages of activities: pre-processing, core business processing and post processing activities.

This paper focuses on making the Business processing activities tier lean and efficient. By leveraging recent advances in technologies, a methodology is described by which the OLTP applications can be successfully transformed into agile systems. The methodology enables a view of the Business Tier in five dimensions. The first dimension is identifying the business's core critical path and how to make it lean. The second dimension is how to enhance concurrency of the activities in the critical path. The third dimension is to improve the parallelism in execution of concurrency. The fourth is to separate I/O operations off the critical path. The fifth being how to minimize contentions for shared resources to ensure higher efficiencies. The paper also presents, the results of the experiments carried out by applying the above recommendation and the performance improvements to decide the optimal setup for an environment for given workload.

Addressing the five dimensions of Business Tier, this paper demonstrates the transformation that can be achieved which will enable the business to be agile and respond to market ecosystem demands in a very efficient and effective manner.

Keywords

layering; concurrency; parallelism; OLTP; CPU affinity; speedup; lean critical path; Architecture; Performance gain;

1. INTRODUCTION

Traditionally, large enterprise applications were running on monolithic main frame based systems. These systems were designed for automating initial business process requirements. As the business expanded and witnessed growth [1][2], these systems were modified to meet the growing needs without changing the underlying architecture. Modifications thus made, unwittingly combined client access, presentation interface, the business processing and database activities into one large monolithic piece. Though the model was simple, it was not scalable and had limitations in flexibility and security. The only way to scale was to adopt distributed computing having multiple installations of an application running at geographically different or same location(s) (Fig. 1), depending on the user distribution. This approach adversely increased the cost and manageability issues.

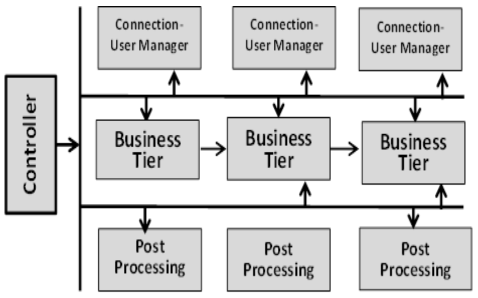

Fig1: Tightly Coupled Distributed System

The next phase of evolution was driven by the advent of Client Server architecture. This enabled applications to be scaled for performance. The clients were thick, combining the presentation and the business logic; and the database was on the central server. The server had higher capacity to support multiple clients. This architecture brought in scalability to some extent and ensured security of the core data; but, was not successful on clients over slower network connections and lacked flexibility to support different types of clients.

The e-commerce growth had paved the way for various types of customers to have direct access to the enterprise systems, customers, business partners, other stake holders and information agencies to have faster access. Each of these users accesses the system through different connectivity mechanisms and IT infrastructure at their end. Any change either functional or technical, had a direct impact on the client software as well as the server software, which increased the time of the release process and therefore time to market. In a two-tier model, managing the access controls and managing the front-end for varying needs with thick clients were becoming serious constraints. With the exponential increase in usage of internets and WANs, e-commerce business applications expanded at a fast pace that demanded the applications to ensure all of the following non-functional requirements as essential to support the business: (i)Scalability (ii) Flexibility (iii) Security (iv) Availability (v) Cost [3].

This resulted in the evolution of multi-tier architecture (Fig. 2). It operates by splitting the application into distinct tiers that could be run on different physical machines. This exploits the right hardware, software platforms for that particular tier.

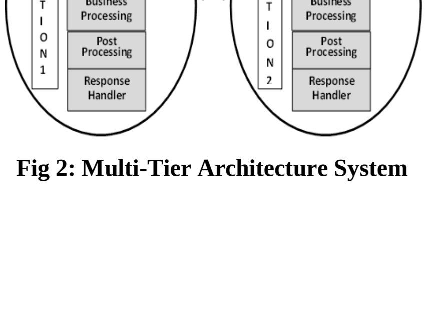

Fig 2: Multi-Tier Architecture System

The 3-tier architecture consisting of presentation tier, business tier and database tier is the most common architecture with a thin client connecting to the enterprise application. Presentation tier is the face to the outside world mainly dealing with stateless data. The other two tiers are responsible for business processing, reliability, recovery and storage of business data [4][5][6].

This paper is focused on improving the scalability and performance of the business tier of such mission critical real- time systems with many connected channels and users transacting online. The key design guidelines and implementation details examined for making the critical path lean and efficient are: Separation of processes, taking I/O off the critical path and thread management for parallelization. In addition, the deployment using CPU affinity and process priorities on multi core hardware are studied and validated with experiments for further scaling.

In summary, our contributions are: (i) Transformation techniques to make critical path lean, (ii) CPU affinity introduction for effective parallelization by minimizing switching overheads (iii) Spinning on shared resources to maximize resource utilization, (iv) Application controlling the I/O by taking I/O off the critical path, and (v) Methodology to derive optimal number of tasks for maximum parallelization gains.

The remainder of the paper proceeds as follows. The next section describes the motivation of our work. Section 3 describes design techniques and methodology, recommended in our proposal. Section 4 describes our experimental methodology and workloads. Section 5 details the scope of experiments conducted to validate the methods proposed in section 3. Section 6 presents its evaluation. Finally, Section 7 presents our conclusions and future work.

2. MOTIVATION

High throughput and low response time expectations of the new business models: High volume mission critical systems are characterized by thousands of users connected simultaneously and generating millions of transactions every second. All of the transactions validated for hygiene and business rules will be forwarded to the Business Tier for actual business processing. The business processing is supposed to deliver high throughput, fast response time along with accuracy, consistency and reliability. The challenge is that the business tier needs to cope up with high degree of concurrency issues arising out of handling state-full data as against the other tiers where majority of the handled data is stateless and naturally paves way for high level of concurrency and parallelism otherwise termed as Embarrassingly Parallel [7].

Scalability and Flexibility demands in changing Business environment: In this era of connected internet world, self-servicing models with thousands of users operating "Anytime Anywhere" concurrently and generating millions of transactions have increased the performance requirements of business critical real-time systems multi-fold. Large, real-time OLTP systems supporting the retail service sectors like financial, banking, telecom and aviation industries are monolithic and are not flexible for business transformations and changes. These systems are mission critical in nature and have been main drivers of business. These systems have evolved over decades and have become a bottleneck for the business in terms of capability to be agile and support market asks. This has resulted in, increasing the risk factor rather than being a business enabler. These organizations are embarked in transforming these OLTP systems into more agile systems using data stream processing and event processing models to address the scalability, throughput, latency and reliability demands of the emerging business ecosystem.

The following is the summary of non-functional architectural considerations of the emerging business environment:

Scalability: In a multi-tier architecture, as the tiers can be hosted on the different machines it naturally brings performance gains. However, the performance of the tier itself can be maximized by choosing the platform that is best suited for a specific tier. Horizontal or vertical scaling can be used depending on the design and nature of data handled; additionally, by fine tuning the coding and deployment of components of the tier. This is further discussed below.

Flexibility: Any change in technology or function local to the tier will not affect other parts of the application. For example, a hardware change for the database can be done without affecting the other parts. Normally, presentation undergoes frequent changes which will be local in this architecture.

Security: The tier-based architecture helps in applying the security rules appropriately. The database which has all the business data should not be exposed to the outside world whereas the presentation tier is the one that connects to the outside world and acts as a first barrier to identify Denial Of Service attacks. This tier therefore does not contain any business logic and is placed in the De Militarized Zone (DMZ). The security policies can hence be applied as per the tier.

Availability: The most common way to ensure high availability and eliminate single point of failure is by redundancy. Multiple instances of a tier can be run to provide high availability in addition to load balancing [8].

3. PROPOSED DESIGNS FOR HIGH PERFORMANCE IN BUSINESS TIER

We now look at the five dimensions individually and apply techniques that will improve leanness and efficiency of the business tier to handle growth. The growth could just be a volume increase in terms of users and transactions or could be a new type of business introducing new type of transactions. The former one requires the system to be scalable and the latter requires the system to be adaptable and flexible to changes without compromising on performance [9].

3.1 Making Critical Path Lean

The critical path decides the least possible time in which the core business transaction in the system can be completed [10]. The key goal of identifying and making the critical path lean is to ensure minimum response time and maximum throughput with reliability. To enable us to do so, we set ourselves two key objectives. (i) Split and group functionally and technically related modules with the aim of being able to deploy them on multiple machines to enhance parallelism and concurrency. (ii) Separate ancillary processing from these to make critical path for core business transactions lean.

Broadly speaking, core Business Tier activities comprises of four major activities: Pre-Processing, Real time Master File updates, External Events processing, and Response processing. Pre-processing performs just-in-time real-time validations, enrichment, filtering of information before handing over to Core Business process. Core Business process performs the actual business transactions on validated data received from pre-processing step. At the time of actual business transactions being performed any external event having a bearing on the business transaction has to be considered as the impact could either accelerate or negate the process step. While the above three activities are being performed, responses to various other systems have to be provided to ensure that all information required for various stakeholders are communicated in a timely manner.

To enable us to make the critical path lean this paper proposes

(i) Separation of Pre-processing
(ii) Separation of Real Time Master Files Updates
(iii) Separation of External Events Processing
(iv) Separation of Response Processing

as described below:

(i) Separation of Pre-processing: The pre-processing task handles receiving of inputs from the preceding tier and performs just-in-time real-time validations, enrichment, filtering, etc., and passes on the incoming data to the next task, the Core Business Process. This is a possible candidate to be separated and can be made concurrent and parallel. This separation helps to validate/pre-process the input and handles the rejections independently; while the business transaction is executed by another process to increase the throughput. With this separation, the Core Business Process performs the actual business transaction only on the validated data received from the earlier process/task.

(ii) Separation of Real Time Master Files Updates: The Core Business Process would have to access the masters for any updates/ validations in the course of the transaction processing. This operation is computation intensive and also involves I/O. It is therefore recommended to handle these updates through a separate process. The same can operate in concurrent and parallel mode. The Core Business Process then can only refer to the updated data and has no overhead of updating/validating these master changes.

(iii) Separation of External Events Processing: In real-time systems, the occurrence of external events has an impact on the Core Business Process. These events will have to be handled by another concurrent process and the results of the events are to be considered while processing the business transaction. In the legacy systems, this event handling is also built in the path of the business transaction processing. In the newer design, it is suggested to separate the external events processing from Core Business Process to be executed concurrently. The Core Business Process will only refer to the results of the external events processed.

(iv) Separation of Response Processing: Mission critical real-time systems are always interfaced with many other modules/ancillary systems for post transaction activities. In the earlier generation systems, even sending the response to the interface systems and end consumer were part of the business transaction processing. In the new design, the processed information can be sent to another process through suitable IPC for further processing. This new process is to pass on the responses to a despatching module independently. The despatching module can then transform and distribute the data to the end consumer modules and interfaces, as well as operate concurrently. In some of the critical systems, there is a need to send intermediate processed data to the interfaces on a real-time basis. The above methodology helps Core Business Process in sending the data to the distributor and to continue with the next transaction processing.

3.2 Enhancing concurrency of the critical path activities

Client Server applications typically use the concept of threads as basic design elements. Threads provided by operating systems help to bring in concurrency and ultimately parallelism on multi-processor systems. Mission critical high volume applications are all based on concurrent algorithms implemented using threads. This means that the application has two or three threads that are being swapped in and out by the operating system on a processor. These threads will be 'in-progress' each in the middle of its execution at the same time. When there are multiple cores available, each thread can be assigned to a different core and can execute simultaneously. Hence execution in 'Parallel' is a subset of 'Concurrency'. There are various design models available for implementation of concurrency. The pipeline and master-slave are the most commonly used models in high volume commercial applications [11]. The true simultaneous execution on multi-core platform poses new challenges. Some of the new challenges that may have to be addressed while designing concurrent algorithms for efficient execution on multi-core platforms are: (i) Contentions and concurrency bugs that were encountered on a multiple CPU, (ii) Effect of priorities associated with threads executing on different CPUs, (iii) overheads associated with the OS migrating the threads to less idle processors [12].

(i) Contentions Due to Threads: With the increase in number of threads that execute simultaneously, contention for shared resources correspondingly increases. Increase in contention for shared resources results in degradation of performance due to wait time. To have bounded contention and reduce overheads due to creation and retiring of the threads, it is recommended to maintain a thread pool catering to the functions rather than having each thread for a user/connection. The design of the thread pool and allocating a thread for incoming requests requires careful analysis as the business tier deals with state-full data and ""ordering"" of transactions in some cases would have to be maintained [13][14].

(ii) Effects of Priority across Cores: The thread priorities can behave differently on multi-core. In a single core machine, the developer can assume and optimize for performance that the thread of lower priority will not be scheduled on the processor when a higher priority thread is ready and the higher priority thread will not be interrupted. But in multi-core when the two threads run on two different cores both could run simultaneously. Therefore such an assumption cannot be made while programming or migrating to multi core [12].

(iii) Migration of Threads across Cores: In multi core systems, OS scheduler executes the task of assigning the processes/threads to the CPU core and migrates the same in case of imbalances created due to some CPUs being lesser used. In the process of doing so, the scheduler maintains ‘soft affinity’ so that threads that share resources are not scheduled on different cores. Every migration has switching overheads and therefore critical processes should be running on the same core for high performance [15]. Towards retaining critical processes in the same core, OS provides facility for CPU affinity by which developer can specify the processor to which a critical thread or process can be bound. The CPU binding requirements need to be analysed by the designer carefully for high performance of critical threads [16].

3.3 Enhancing Parallelism

Once the tiers are decomposed and concurrency introduced to enhance the performance, the same can be further improved by introduction of parallelism by deploying these separated processes in dedicated cores of a multi core multi CPU machine. The optimal deployment and parallelization gains are further demonstrated in our experiment and results evaluation.

Deployment of Transformed Business Tier (Fig. 4) with CPU Affinity: The communication between cores on the same CPU has less latency because the cores communicate more quickly as they are all on the same chip. The proximity of multiple CPU cores on the same socket allows the cache coherency circuitry to operate at a much higher clock rate than is possible if the signals have to travel off-chip [17].

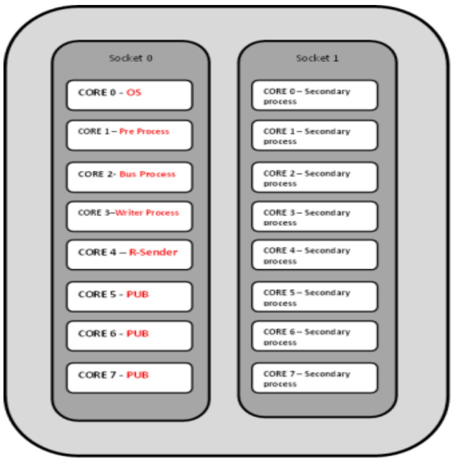
Socket 0
CORE 0 - OS
CORE 1 – Pre Process
CORE 2- Bus Process
CORE 3–Writer Process
CORE 4 – R-Sender
CORE 5 - PUB
CORE 6 - PUB
CORE 7 - PUB
Socket 1
CORE 0 – Secondary process
CORE 1 – Secondary process
CORE 2 – Secondary process
CORE 3 – Secondary process
CORE 4 – Secondary process
CORE 5 – Secondary process
CORE 6 – Secondary process
CORE 7 – Secondary process

Fig 3: Transformed Business Tier Deployment with CPU Affinity

Considering the above aspect the processes that are in the critical path of the transformed business Tier in Fig. 4, are recommended for deployment on the cores of the same CPU to have minimum communication lag towards faster response time for transactions. Other non-critical ancillary processes that could be large in number are deployed on the cores of the second CPU For example, the transformed business tier shown in Fig. 4can be deployed in 2-CPU, 8 core each, as show in the Fig. 3.

3.4 Maximizing Parallelization

Traditionally, computer software has been written for sequential processing using procedural languages, oriented towards single CPU. However, with advent of multi-processor/multi-core platforms parallel execution became a reality and the applications need to be re-architected. Parallelization is an optimization technique with a goal to reduce time executing on multiple processors or cores. The computation is such that a large problem is divided into smaller ones using models such as pipelining and master-slave, that can execute concurrently and the order of their execution is not important. But there would be some activities that are un-parallelizable forming the serial component of workload, resulting in sequential processing.

To address resources contention, it is pertinent to minimize serial components and maximize parallelization of different processes in the critical path to improve the performance of the system.

(i) Optimal Length for Critical Sections: Length of the critical section determines the predictability of response time. Too many short critical sections introduce context switching overheads and too long critical sections result in increased serial processing affecting the performance [18][19]. The code should be written in native language for best performance and not a wrapped up piece that introduces execution overheads [20].

(ii) Minimisation of Contention with IPC : Across concurrent processes the interaction models that are commonly used are shared memory and point-to-point queues. To avoid contention it is recommended to use queues over shared memory [21][22]. In a high volume system, shared memory results in contention and therefore inhibits scaling. Whereas, queue implementations have the advantage of locality of reference and there is no contention between readers and writers as they can work on different areas of a queue implementation. To achieve best results, it is suggested to design with a single writer to a queue.

(iii) Minimization of Contention for Shared Resources: Contentions increase the length of the time spent in critical path as they serialize the activities. Shared resources are commonly used in all real-time high volume applications. With the new hardware platforms that support multi-processing capabilities the new challenge posed is the increased contention for shared resources. Minimizing contention is a crucial factor towards reaping the benefits of high performance multi-core hardware.

To manage the shared resources for consistency, the mutually exclusive locks are used. The performance of mutually exclusive locks in parallel applications degrades significantly on time-slicing multitasking systems due to the pre-emption of processes holding locks. Other processes busy-waiting on the lock are then unable to perform useful work until the pre-empted process is rescheduled and subsequently releases the lock [23]. The mutual exclusion methodology therefore, needs to be properly chosen. Though blocking ensures robustness, there is a cost to it and results in very bad outliers [24]. In a multi-core platform, where a CPU can be dedicated to a process to avoid context switching overheads, non-blocking methods such as atomic spinning for a flag is recommended as there is no idle time as soon as the shared resource becomes available [25]. When a CPU is dedicated for a process the condition of non-blocking resulting in indirect priority inversion due to CPU starvation for other processes does not arise.

(iv) Data Management for High Throughput and Fast Response: Complex applications require the use of data storage in a relational database for data processing. The traditional database management systems are disk oriented and designed with a view to optimize disk operations. In the 3-tier architecture the business tier does the transaction processing and queries/updates to database. The enterprise database is not ideally suited for such high volume mission critical real-time applications [26]. The slowest operation in the whole processing chain is the writing and reading from the database which becomes a serious bottleneck for the performance of the business tier as the number of users/transactions increase.

High volume mission critical systems, therefore adopt embedded in-memory database that offers high throughput, faster response by bringing the data close to the application, small footprint to reside in memory and provide a bridge between the business tier and database tier. Depending on the performance requirements the in-memory DB could be custom database or standard products like Times Ten [26]. At run time for faster throughput the database, all of user data, indexes and system catalogues, would reside completely in-memory and is persisted to disk for the ability to recover and restart.

3.5 Taking I/O off the Critical Path

The critical path normally includes journaling of certain transactions data for recovery and future references and logging for the purpose of audit/errors. The I/O activity in the critical path, significantly delays the processing. In high volume systems, delay in processing of a single data set not only affects that transaction, it also delays all the further incoming transactions. In mission critical systems as the fast recovery of the logged information is crucial, the critical logging cannot be avoided left only to be basic core transaction and some of the intermediate processed data is also important. Therefore, the methods suggested are (i) asynchronous commit or (ii) having a separate daemon process that handles all the I/O requests from the business processes independently [27].

4. TRANSFORMED ARCHITECTURE

The transformed Business Tier after applying the above principles is shown in (Fig. 4) along with lean critical path. On the left hand side of Fig. 4. is the monolithic system with all the functional components in a single process. Post transformation the connection manager of the earlier system is mapped to connection tier of transformed architecture; concurrency and parallelism is introduced in this tier to manage the various sources of incoming data streams. This is possible as the incoming data is stateless at this point.

Circled portion is the transformed critical path of the business tier and the same is mapped with the hygiene validation and business processing. It is to be noted that the business tier is broken into Pre-Process, Core Business Process and Post Process layers. In pre-process the enriching, housekeeping and external event based interventions are taken off the critical path and put in parallel to the critical path. In the core business process in-memory database is used to make it lean and efficient.

I/O is taken off the critical path in asynchronous mode in a separate process ‘Safe Store Writer’. On the outbound, response sender uses multiple threads and publisher in concurrent mode to send the packets on the network to the N end consumers.

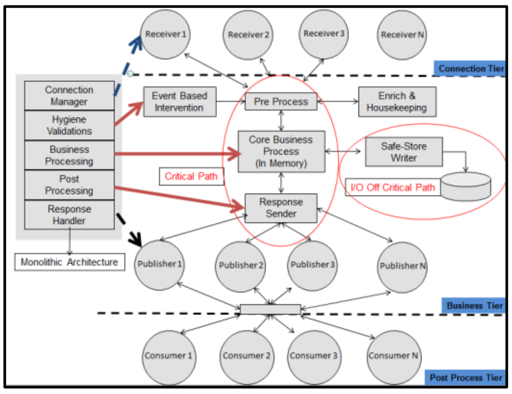

Fig 4: Transformed Business Tier

5. PERFORMANCE EXPERIMENTS

The experiments are carried out for the “critical path” post the separation of tiers to examine the possible improvements by adopting the key improvement suggestions discussed in section 3. The following three techniques are used in the experiments to demonstrate the performance impact in developing a lean and efficient “Business Tier” in a data stream application; (1) Study of CPU affinity and spinning on the queues in the modern multi core hardware, (2) Study of concurrency and parallelism in the tasks along with CPU affinity and spinning and (3) Study of application controlling the I/O operation by taking I/O off the critical path to avoid jitter and improve performance. The subsections describe the experimental setup and the details of the experiment runs for measuring the impact of the above 3 techniques for scaling the performance.

5.1 Experimental Setup

Functional Setup: The experimental setup of the “critical path” taken up for analysis is as shown in Fig. 4. A typical transaction flow is used for the study i.e., Pre-Process is taking care of routine housekeeping, filtering, enrichment, master updates and last minute dynamic validation if any; besides these, the possible parallel static stateless validations that may be carried out before reaching the business tier. Core Business Process does the actual business transaction processing taking into account the effect of the external events, update/processing of the incoming transactions and essential I/O that may be needed for recovery in a real-time mission critical applications. Response Sender is the process responsible for sending the output to multiple receivers and interfaces over network. The same can be parallelized for performance using multiple senders as this stage is normally stateless. This flow involves CPU bound activities, memory bound activities and I/O bound activities.

Hardware Setup: The study runs were executed with a 2 CPU Xeon Intel processor with 8 Core each running linux6.x operating system. The CPU 0 was used for the critical path services and the CPU 1 was used for all other ancillary services. Within CPU 0, Core 0 was assigned to OS; Cores 1-3 were used for “Receiver” and “Business Processor” and 4-7 were used for “R-Service” components.

5.2 Methodology used for analysis

In this subsection we define the metric and laws we use to measure the performance of the resultant architecture in the experiment setup post applying the techniques defined in section 3. We then describe in subsection C the 3 different techniques we study in the experimental setup. Amdahl and Gustafson's law is used to understand the speed up behavior in a multi-core environment setup [28]. This will give a measure to optimize the critical path further by minimizing serial components and maximizing parallelization and arrive at an optimal deployment setup for high performance.

5.3 Scenarios for Experimental studies

(i) CPU Affinity and Spinning on the Queues

In this experiment, we measure the performance improvement with CPU affinity and impact of spinning on IPC queues in the “critical path”. The first run is done without any CPU affinity and using event notification on the IPC queues. In the subsequent runs the critical path processes are assigned individual core within a CPU socket and all the other ancillary processes assigned in a different CPU socket and cores as in Fig. 4. This is to take advantage of multi CPU and multi core new generation hardware to demonstrate the throughput improvement along with latency reduction. In addition, spinning on the queue is introduced instead of event notification at process level as dedicated cores are allocated to the processes in the “critical path”.

(ii) Taking I/O off the Critical Path and Optimization

In this experiment, we measure the improvement of performance by the application controlling the I/O operation by taking I/O off the critical path. The first run is carried out with I/O in the transaction path. The throughput and latency is recorded. These readings are compared with the subsequent runs in which I/O operation of logging various activities is taken off the critical path. Also, optimizations such as functional in lining, context localization and contention minimization are introduced to make the critical path efficient.

(iii) Non-blocking, Concurrency and Parallelism
In the second set of experiments, we measure the performance improvement of multiple publishers deployed on multiple cores (2-6 publishers) as against single publisher in the "Response Sender" process in the outbound. This is to study the impact of introducing concurrency and parallelism for maximizing the CPU cores usage. Non-blocking on multi-core platforms is tested for further improvement. The results are tabulated and interpretations are discussed in the next section of this paper to demonstrate the performance improvement.

(iv) Workload Variations
The above experiments are repeated for varied workloads of incoming arrival of packets at 20K/sec, 40K/sec and 60K/sec. This is done to verify the impact of increasing workload on effective parallelization in the experimental setup.

6. EVALUATION

6.1 Lean Critical Path with CPU Affinity and Varying Workloads

The experiments are performed using the experimental setup and workloads outlined in section 5. Augmentation of hardware and software resources is normally adopted to improve the performance, with the assumption that the critical path is independent of the workload. However, in systems with parallel operating components, this assumption leads to incorrect results as the behavior of the critical path varies under different workloads due to serial and parallelization impact and parallelization overheads. This establishes the fact that simple extrapolation of performance under scaled down loads will not work for expected load.

The details of the performance experiments carried out to understand the impact of introducing CPU affinity in any typical data stream processing application is discussed in this subsection. First, Fig. 5(a) shows the response time measured for processes in the "critical path" with no CPU affinity and with CPU affinity for 93% and 95% of packets processed.

Impact of CPU Affinity				
% of Data Packets	93%	95%	99%	100%
	Response Time in µs			
With No CPU Affinity	100	150	8650	413200
With CPU Affinity	48	49	180	237

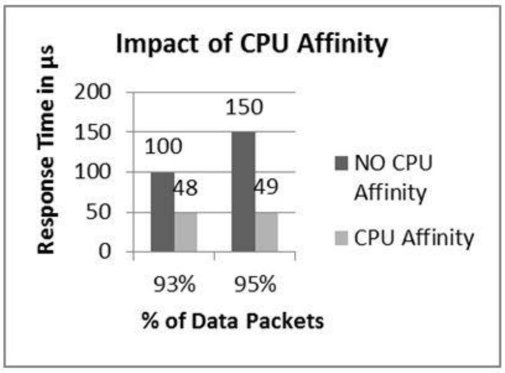

Fig 5(a): CPU Affinity

Fig. 5(a) shows that introduction of CPU affinity for processes in the "critical path" using multi core hardware, improves the response time. In this case, the response time improvement is 52µs and 101µs for 93% and 95% of the packets processed. A significant improvement of 52% - 67% is observed in response time. At 99-100% the recorded data shows multifold improvement in latency and reduction in the outlier packets.

6.2 Enhancing Parallelization of critical path

Impact of CPU Affinity with Variation in Load				
% of Data Packets	93%	95%	99%	100%
Variation Load	**Response Time in µs**			
@20K	48	49	180	237
@40K	63	70	314	290
@60K	78	90	333	538
Without CPU Affinity	100	150	8650	413200

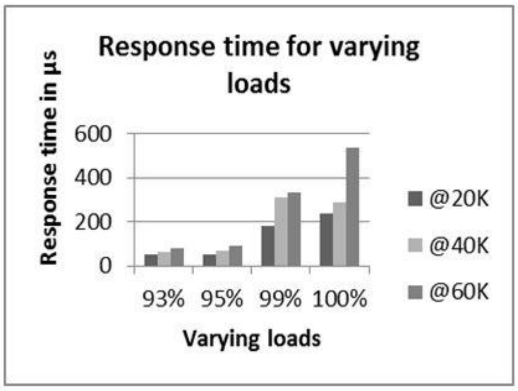

Fig 5(b): Variation in Arrival Rate

Fig. 5(b) shows the impact of varying workloads on the same setup. Clearly, as the input rate is increased to 40K messages (problem size is bigger), performance gains start reducing. The data shows a reduction in the response time gain by 15µs and 31µs respectively for the 93% and 95% of packets processed as against the earlier response gains of 52µs and 101µs. At 60K throughput, a further reduction in performance gain is observed. Due to increase in parallelization overheads with the increase in arrival rate, parallelization gains show a reduction.

In summary, the introduction of CPU affinity recommended in the study results in considerable performance gains because of the parallelization benefits. However, when the workload increases, the gains would reduce as seen in the experiments. To enhance parallelization, it is necessary to analyze the hardware and software setup for the deployment, keeping in mind the anticipated production workloads to establish predictable performance.

6.3 Maximization of Parallelization

Further experiment runs were conducted to analyse the effect of number of parallel and serial tasks on the performance in a multi core machine. With CPU binding the number of parallel publisher process threads in the sender process was varied to verify the effect of parallelization and parallelization overheads on performance. The objective is to determine the optimal number of parallel publisher process threads that would improve the performance.

The performance experiment runs were carried out for 1 - 6 publisher process threads with 3 cores dedicated for the publisher process threads. In addition, the 6 publisher thread run was repeated with 6 CPU cores dedicated for the publisher threads. Table 1 gives the details of the response time for 90%, 96% and 99% of packets processed for the different runs. The arrival rate is randomized and kept bursty in nature to reflect the real life scenario. Using the 99% data in Table 1, the results tabulated in Table 2 are derived to study the impact of serial and parallel components.

Table 1.Results of Variation of Number of Publishers

	Bursty Input--1 million - with Number of Publishers Variation						
Messages	**1Pub**	**2 Pubs**	**3 Pubs**	**4 Pubs**	**5 Pubs**	**6 Pubs - 3C**	**6 Pubs-6C**
90%		**37.7867**	**36.7530**	**34.3507**	**36.3660**	**45.4211**	**40.6145**
96%	**67.6631**	**46.5872**	**45.4589**	**40.9507**	**41.7257**	**56.7850**	**49.4165**
99%	**97.5813**	**61.2239**	**59.5325**	**56.4405**	**56.9371**	**72.3974**	**64.0004**

Table 2.Analysis of 99% Data

Analysis of 99% level Data for Parallelisation Impact					
Publishers	**Response Time in µs**	**Speed Up**	**Amdahl's Speed up**	**Serial effect**	**Parallel effect**
1	**97.5813**		**1**		
2	**61.2239**	**1.5938**	**1.5957**	**44.00%**	**56.00%**
3	**59.5325**	**1.6391**	**1.6411**	**41.40%**	**58.60%**
4	**56.4405**	**1.7289**	**1.7309**	**36.66%**	**63.34%**
5	**56.9371**	**1.7138**	**1.7176**	**37.33%**	**62.67%**
6	**72.3974**	**1.3479**	**1.3538**	**60.80%**	**39.20%**
6	**64.0004**	**1.5247**	**1.5267**	**58.60%**	**41.40%**

In Table 2, the actual speedup is computed by dividing the response time data of single (1) publisher for 99% by the data of 2, 3, 4, 5, 6 publishers using data in Table 1. By substituting in Amdahl’s Law, the value of actual speedup, the % of serial and parallel processing on an incoming packet in the data stream processing setup given in Fig. 4 is derived. It is observed that the speedup improvement is based on the proportion of serial and parallelization effect in the process path. It is observed from the data in Table 2, by increasing number of publishers the parallel effect improves up to 4 publishers. After this, due to increasing parallelization overheads and contentions, the speedup reduces.

In the experiments with 6 publishers, the first experiment is with 3 cores assigned for the publishers and the second experiment is with 6 cores assigned. This is done to verify the benefit of further parallelization with additional CPU cores introduced. From the results it is observed, though there is an improvement over the first experiment, it is not linearly giving increased performance because of parallelization overheads. This is experiment is clearly demonstrating the impact of the parallelization overheads.

Table 3.Analysis of 96% Data

Analysis of 96% level Data for Parallelisation Impact					
Publishers	Response Time in µs	Speed Up	Amdahl's	Serial effect	Parallel effect
1	67.6631		1		
2	46.5872	1.4524	1.4535	53.20%	46.80%
3	45.4589	1.4884	1.4897	50.69%	49.31%
4	40.9507	1.6523	1.6536	40.71%	59.29%
5	41.7257	1.6216	1.6246	42.33%	57.67%
6	56.7850	1.1916	1.1957	75.45%	24.55%
6	49.4165	1.3692	1.3714	67.50%	32.50%

Table 4.Analysis of 96% Data

Analysis of 90% level Data for Parallelisation Impact					
Publishers	Response Time in µs	Speed Up	Amdahl's	Serial effect	Parallel effect
1	45.5631		1		
2	37.7867	1.2058	1.2063	74.35%	25.65%
3	36.7530	1.2397	1.2402	70.95%	29.05%
4	34.3507	1.3264	1.3270	63.04%	36.96%
5	36.3660	1.2529	1.2547	69.55%	30.45%
6	45.4211	1.0031	1.0064	99.05%	0.95%
6	40.6145	1.1218	1.1246	86.70%	13.30%

In Table 3 and Table 4 the actual speedup is calculated for the 96% and 90% using the experimental data in Table 1, the serial and parallelization quantum is computed as in Table 2 using Amdahl’s law. The data indicates that at 96% and 90% levels, the serial effect is more and parallelization effect is lesser. This is on account of the velocity of the incoming packets at 96% and 90% level being less compared to the 99%. As the velocity reduces, use of parallelism reduces as the data gets processed before the arrival of the next packet due to sufficient inter-packet delay. At 96% and 99% there is a relative improvement in the use of parallelism and increased parallelization effect is seen. This is due to the fact as the problem size increases, effect of the serial portion is getting diminished and the parallelization with its overheads increases, as defined by Gustafson’s law.

Table 5.Analysis of Data for Parallelization Impact @40K Rate

Analysis of Data for Parallelisation Impact @40K Rate					
Publishers	Response Time in µs	Speed Up	Amdahl's	Serial effect	Parallel effect
1	131.894	1.000	1.000		
2	69.978	1.885	1.884	29.60%	70.40%
3	50.146	2.630	2.632	7.00%	93.00%
6	68.579	1.923	1.923	28.00%	72.00%

Table 6.Analysis of Data for Parallelization Impact @Bursty Rate

Analysis of Data for Parallelisation Impact @Bursty Rate					
Publishers	**Response Time in µs**	**Speed Up**	**Amdahl's**	**Serial effect**	**Parallel effect**
1	**97.6631**	**1.0000**	**1**		
2	**56.5872**	**1.7259**	**1.7261**	**36.90%**	**63.10%**
3	**45.4589**	**2.1484**	**2.1493**	**19.79%**	**80.21%**
6	**70.6665**	**1.3820**	**1.3825**	**58.50%**	**41.50%**

Table 5 and Table 6 give the results of experiments carried out for 40K/sec arrival rate and bursty random arrival rates. The results show that at the higher arrival rate of 40K/sec the serialization effect is seen reduced and parallelization is seen increased. In the bursty random rate as the arrival rate is less compared to that of 40K/sec the packets get processed before the arrival of the next packet and hence the serial portion is seen higher and parallelization effect is seen less. This confirms the behavior of the same in line with the earlier experiment.

In summary, in a multicore implementation it is essential the designer looks for opportunities for parallelization in the critical path. The quantum of serial portion and parallelization will determine the amount of Speedup. Also, at a higher velocity of arrival the serial portion diminishes and increases the parallelization resulting in better speedup at these levels.

6.4 Controlling the I/O by application and Optimizations

In addition to the recommendations in the previous section, I/O is taken off the critical path and other optimizations. The improvement in response time and throughput are shown in Table and Fig. 6(a) and 6(b).

Table 6 (a) application controlling the I/O operation

Process Wise Improvement with I/O OFF		
	Latency in µs	
Processes	**Before**	**After**
Pre-Process	**3**	**1.8**
Core Transaction	**30**	**18**
R-Process	**23**	**11.9**
R-Sender	**13**	**8.5**
R-Publisher	**17**	**13.7**

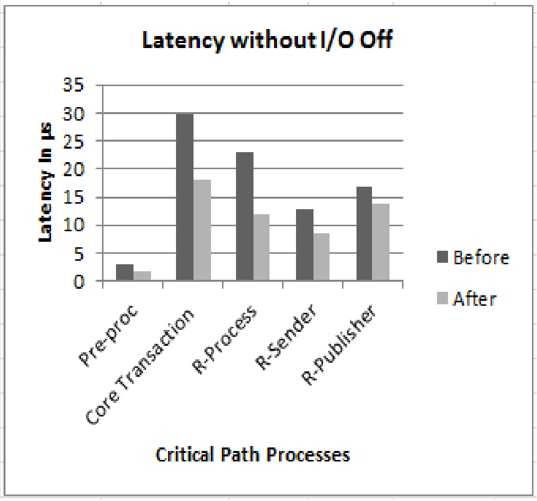

Fig 6(b) application controlling the I/O operation

Table 6 (b) application controlling the I/O operation

Process Wise Improvement with With I/O OFF		
	Throughput K/Sec	
Processes	**Before**	**After**
Pre-proc	**333**	**556**
Core Transaction	**33**	**56**
R-Process	**43**	**84**
R-Sender	**77**	**118**
R-Publisher	**59**	**73**

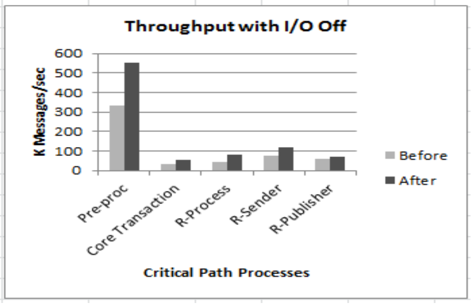

Fig 6(b) application controlling the I/O operation

The experimental results show the improvement of performance by the application controlling the I/O operation by taking I/O off the critical path. The impact of reduced latency and increased throughput is clearly seen in all these processes of the critical path from the Fig. 6(a) and 6(b). In the core transaction process the latency has improved from 33µs to 18µs and the throughput improved from 33K to 56K messages per second. There is a 40% improvement in latency and 41% improvement in throughput. Similar improvement is observed in all other processes.

In summary, the recommendation of application controlling the I/O operation helps in controlling the jitters and the outliers impacting the latency and throughput heavily.

7. CONCLUSION

In this paper we have defined the methodology to be used to transform OLTP legacy applications to agile applications by introduction of a three tiered architecture of Pre-processing Tier, Business Tier, Post processing tier. We have further demonstrated the key factors to be considered to enable the critical path of the Business Tier lean and efficient. Making the critical path lean and efficient, thereby enabling high throughput and low latency, using the following: (i) Separation of the processes within a tier and introduction of CPU affinity in a multi core hardware; a significant improvement of 52% - 67% is observed in response time and at 99-100% the recorded data shows an exponential improvement in latency and reduction in the outlier packets observed in the experimental evaluations. (ii) Introduction of optimal concurrency and then enhancing and further maximizing parallelization of tasks for the best possible performance gains is to use up to 4 publishers. Beyond 4 publishers, due to increasing parallelization overheads and contentions, the speedup reduces (iii) making the Application control the I/O operations by taking them off the critical path to reduce jitter and minimize the outliers; there is a 40% improvement in latency and 41% improvement in throughput in the core transaction processing. Similar improvement is observed in all other processes.

We recommend that this kind of architectural analysis and approach is required for achieving extremely high performance, and such experiments need to be carried out for identifying the optimal levels of parallelization within a critical path considering the workload for the best possible speedup in mission critical real time application transformations.

In this paper, we have focused on optimizing the “Business Process Layer” in a typical “Data Stream Processing” application. The principles examined in this paper can be applied more generally to further enhance the performance of any component of a “Data Stream Processing” application in transforming mission critical real time applications for high performance. The other layers such as (i) Pre-Process, (ii) I/O in the Critical Path, (iii) Response Delivery and (iv) in-memory database also require similar kind of analysis using the five dimensions used in this study, and the same can be taken up as future work items.

8. ACKNOWLEDGEMENTS

The authors would like to thank the support of National Stock Exchange of India Limited for the lab facilities for the research work.

9. REFERENCES

[1] EDS, “Financial services legacy modernization,” tech. rep., EDS, 2007.
[2] L. Wu, H. Sahraoui, and P. Valtchev, “Coping with legacy system migration complexity,” in Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCSŠ05), June. 2005, pp. 600-609.
[3] IBM, Building Multi-Tier Scenarios for Web Sphere Enterprise Applications, I. Redbook, Ed. IBM Redbooks, 2003.
[4] C. Michiels, M. Snoeck, W. Lemahieu, F. Goethals, and G. Dedene, A Layered Architecture Sustaining Model Driven and even driven software development, Springer, Ed. 5th International Andrei Ershov Memorial Conference, PSI 2003, July. 2003, pp. 58-65, vol. 34, no. 4.
[5] H. R. Simpson, “Layered architecture(s): Principles and practice in concurrent and distributed systems,” in Engineering of Computer-Based Systems, 1997. Proceedings, International Conference and Workshop, March. 1997, pp. 339–350.
[6] R. Peacock, “Distributed architecture technologies,” IT PRO, June. 2000.
[7] R. Choy and A. Edelman, “Parallel mat lab: Doing it right,” in Proceedings of the IEEE (Volume: 93, Issue: 2), 2005.
[8] IBM, “Building multi-tier scenarios for web sphere enterprise applications,” IBM Redbook, Tech. Rep., 2003.
[9] Tune, E., La Jolla, Dongning L, Tullsen, D.M. and Calder, B., 2001, “Dynamic prediction of critical path instructions”, High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium, Jan. 2001, pp. 185-195.
[10] Shripad Agashe, “Predictive analysis using critical path method & Amdahl’s law,” cmg.org, 2010.
[11] S. Kleiman, D. Shah, and B. Smaalders, Programming with threads. Prentice Hall, 1996.
[12] S. Akhter and J. Roberts, Multi-Core Programming, I. Press, Ed. Intel, 2006.
[13] F. R. Johnson, R.Stoica, A. Allamakee, and T.C. Mowry, “Decoupling contention management from scheduling,” in ASPLOS XV Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems, ACM, New york, March. 2010, pp. 117–128.
[14] D. Kimpe, D. Carns, P. H, Harms, K, Wozniak, J.M, Lang, S, and R. B. Ross, “AESOP: expressing concurrency in high-performance system software,” in Networking, Architecture and Storage (NAS), June. 2012, pp. 303-312.
[15] Black, D.L, “Scheduling support for concurrency and parallelism in the Mach operating system,” IEEE Computer, 2002.
[16] M. Rajagopalan, B. T. Lewis, and T. A. Anderson, “Thread scheduling for multi-core platforms,” Article 2. Usenix, 2007.
[17] G. K. Lockwood, “Processor affinity,” tech. rep., glennklockwood.com, 2014.
[18] R. Johnson, I Pandis, and A Ailamaki, “Critical sections: re-emerging scalability concerns for database storage engines,” in DaMoN ’08 Proceedings of the 4th international workshop on Data management on new hardware, June. 2008, pp. 35-40.
[19] M. A. Suleman, O. Mutlu,, and M.K. Qureshi, “Accelerating critical section execution with asymmetric multi-core architectures,” in ACM SIGARCH Computer, March. 2009, pp. 253-264.
[20] F. R. Johnson, “Scalable storage managers for the multicore era,” Ph.D. dissertation, Carnegie Mellon University, 2010.
[21] T. J. LeBlanc and E. P. Markatos, “Shared memory vs. message passing in shared-memory multiprocessors,” in Fourth IEEE Symposium, December. 1992, pp. 254-263.
[22] D. Kranz, K. Johnson, A. Agarwal, J. Kubiatowicz, and B.-H. Lim, “Integrating message-passing and shared-memory: Early experience,” in PPOPP ’93 Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, July. 1993, pp. 54-63.
[23] Maged M. Michael and Michael L. Scott, “Non-blocking algorithms and pre-emption-safe locking on multiprogrammed shared memory multiprocessors,”

Journal of Parallel and Distributed Computing, Volume 51, Issue 1, May. 1998, pp. 1-26.

[24] L. Boguslavsky, K. Harzallah, A. Kreinen, K. Sevcik, and A. Vainshtein, “Optimal strategies for spinning and blocking,” Journal of Parallel and Distributed Computing Volume 21, Issue 2, May. 1994, pp. 246-254.

[25] R. Johnson, M. R. Stoica, and A. natassa@epfl.ch, “A new look at the roles of spinning and blocking,” in DaMoN ’09 Proceedings of the Fifth International Workshop on Data Management on New Hardware, 2009, pp. 21-26.

[26] Times Ten Team, “High-performance and scalability through application tier, in-memory data management,” in VLDB ’00 Proceedings of the 26th International Conference on Very Large Data Bases, 2000, pp. 677-680.

[27] R. Johnson, I. Pandis, R. Stoica, M. Athanassoulis, and A. Ailamaki, “Aether: A scalable approach to logging,” VLDB Endowment, Volume 3, Issue 1-2, September. 2010, pp. 681-692.

[28] M. Gillespie, “Amdahl’s law, Gustafson’s trend, and the performance limits of parallel applications,” tech. rep., Intel, 2008.