Analysis of Application Partitioning for Massively Multiplayer Mobile Gaming

Madan Kumar M. M,
Amit Thawani*, Sridhar V.
Applied Research Group, Satyam Computer Service Ltd, 3rd floor SID Block, IISc Campus,
Bangalore, INDIA 560012
{Madan_MM, Amit_Thawani, Sridhar}@satyam.com

Prof. Y. N. Srikant
Department of Computer Science and Automation,
IISc Campus,
Bangalore, INDIA 560012
srikant@csa.iisc.ernet.in

ABSTRACT
Mobile devices offer the opportunity to play games anywhere anytime. Moreover, networked games allow individual players to interact with other people and to participate in a larger gaming world. Improved network and upgraded software's available on mobiles have given enough scope for massively multiplayer mobile games. An inherent problem is efficient utilization of resources when a numbers of people are playing games in real time. Gaming infrastructure mostly involves gaming servers and it is likely for a gaming server to run short of resources under peak load conditions resulting in degradation of game play. Under this situation possible solutions would be to replicate server to handle more load, increasing the bandwidth, or to maintain different connections with other servers. Since load on server is not likely to happen often, replicating of server infrastructure prove to be costly. To handle such situation possible solution is to partition the game application and off load some of the processing onto either client/server depending on the availability of resources provided the other has sufficient processing bandwidth available. In this paper we address the problem of providing a good gaming experience on mobile devices when server is short of resources. Our approach considers the game which follows client server model and is based on partitioning the game application. We model the game and represent the game as a graph and partitioning game application problem reduces to graph partitioning approach.

General Terms
Algorithms, Management, Measurement, Documentation, Performance, Design, Economics, Reliability, Experimentation

Keywords
Mobile Gaming, Scalability, Application Partitioning

1. INTRODUCTION
Game developers, providers and players get more and more interested in games that can be played everywhere. Multiplayer games allow more people to play together or against each other in the same game. Common examples of multiplayer games are Quake, Doom, and Ever Quest adventures. Massively Multiplayer Game (MMG) is a computer game which is capable of supporting hundreds or thousands of players simultaneously. Typically, this type of game is played in a giant persistent world. MMGs can enable players to compete with and against each other on a grand scale, and sometimes to interact meaningfully with people around the world.

Quake is a popular multiplayer game run by a single server [1], [2]. It can be divided into a server part and a client part. The client part is in charge of graphics and the user controls (Keyboard, mouse, game pad, etc.). The server updates the world states and feeds this information back to the client for it to redraw the graphics accordingly. All network communications are over User Datagram Protocol (UDP). All game logic and physics are carried out on the server and on client it is essentially a graphics-rendering and client-input engine. The client continuously sends packets to the server with the state of the user key presses and mouse position. The server keeps all game state information and sends clients updated positions and appearances of entities in the game world 10 times per second, or every 100 ms to make motion appear smooth. In moments of unusually high latency, the client also attempts to predict the contents of the next sever update, but this is error-prone and considered a last resort.

1.1 Introduction to Gaming on Mobile Device
Recent innovation in mobile games includes distributed 3D graphics applications [3]. Mobile devices offer the opportunity to play games nearly everywhere. Currently mobile devices are designed to support multimedia capabilities. Yet, in order to support games on mobile devices some challenges, which are due to the nature of wireless networks, have to be overcome. Multiplayer mobile game functionality is achieved, through: Infrared, Bluetooth, GPRS, 3G and WiFi. These networks are designed to deliver broadband multimedia and real-time data. The main restrictions a mobile device imposes on graphics and artistic freedom are the available memory size, the processor capacity and most obviously, the limited display size and resolution. Additionally, because of the mobility of devices,

MOBILWARE 2008, February 13-15, Innsbruck, Austria

DOI 10.4108/ICST.MOBILWARE2008.2875

locations can change frequently and introduce several problems due to which game might run slow or device freezes, which is not a good gaming experience. Today technology is heading towards devices which provide good memory, processor capacity and more space for display with better resolution [4]. Challenges for this would be to create concepts and software's that will handle mobile network latency, and to choose right multiplayer platform which must be open enough for future improvements without imposing any restrictions. Game plays should be well balanced for synchronous and asynchronous multiplayer games.

Major challenges with massively multiplayer mobile gaming are:

Latency - The foremost and most critical problem for a real-time game is the network's latency, the time between sending a request and receiving a response. The amount of time it takes to get data from one machine to another and to receive a response is an issue with which all connected games struggle. The available bandwidth in mobile networks is usually lower than in fixed networks. The available bandwidth in wireless networks is dependent on network technology, radio conditions, and subscriber Quality-of-Service (QoS) profiles.

Mobility – Frequent movement of the device form one place to other might change the network unpredictably resulting in broken links and stale routes.

Congestion – In massively multiplayer game, frequent updates does populate the network. To have better gaming experience real-time traffic must arrive in-time even if the network is highly loaded.

Wireless Signal - The mobile signal suffers from fading and interference which gives rise to gray zones and frequent retransmissions. Issues here include the adjustment of the user experience when bandwidth drops or a wireless connection experiences interference and is temporarily out of service. This causes delayed arrival of game updates resulting in loss of game state and game synchronization problem.

Connectivity - Refers to issues related to the connection of the device to a network and other peripherals. These can be wired or wireless connections. The mobile devices are expected to have high degree of perception to maintain the game state updated on time.

Power **-** The use of batteries or power outlets, which modifies the user experience based on the current power usage of the system. In massive gaming environment the need for non exhaustible power supply is important.

Memory - Mobile devices impose major restriction on applications due to limited memory size and display available. For a massively multiplayer gaming environment, need for good memory and processing speed is perquisite. Recent mobile devices from Sony, Nokia have decent internal memory of 64MB and processing speed up to 500MHz. Even though under massive gaming environment the chances for game to run slow or freezing of device is very much possible.

Providing massively multiplayer games on mobile device is mainly dependent on the above mentioned parameters. Improved network and upgraded software's available on mobiles have given enough scope for massively multiplayer mobile games. An inherent problem is efficient utilization of resources when 'n' numbers of people are playing games in real time. In a multiplayer game which follows client server model, a situation where either client or server is running short of resources is a possible scenario.

Having capable mobile device which would support a massively multiplayer game typically having client-server architecture requires a robust and scalable gaming server. A gaming server must be capable enough to handle varying number of players. Having a fixed amount of resources, it is likely for a gaming server to run short of resources under peak load conditions resulting in degradation of game play. Under this situation possible solutions would be to replicate server to handle more load, increasing the bandwidth, to maintain different connections with other servers, etc. Since load on server is not likely to happen often, replicating of server infrastructure proves to be costly. To handle such situation, possible solution is to partition the game application and off load some of the processing onto either client/server depending on the availability of resources provided the other has sufficient processing bandwidth available. For example for the scenario of server running short of resources partitioning will result in off loading some of the processing to connected capable client devices. Partitioning server application to balance load on server by partitioning functions specific to clients such as player collision, player move, etc, would give better performance. To perform partition, important thing to be considered is the availability of bandwidth so that minimal latency is experienced without hampering the game performance.

In this paper we address the problem of providing a good gaming experience on mobile devices when server is short of resources. Our approach considers the game which follows client server model and is based on partitioning the game application. We model the game and represent the game as a graph and partitioning game application problem reduces to graph partitioning approach. We use KL algorithm for graph partitioning to determine client/server specific game logic to be off loaded past partitioning for cases when either of them are short of resources.

2. Related Work

In past, the process of application partitioning is the problem of dividing sections of application code among a set of processors for execution in parallel taking into account the communication overhead between the processors. Application code partitioning of large amounts of code onto numerous processors requires variations to the classical partitioning algorithms, in part due to the memory and time requirements to partition a large set of data, but also due to the nature of the target machine and multiple constraints imposed by its architectural features. Partitioning code for massively parallel machines by Sun System gives an overview of partitioning. Applications which are component based require components to be identified for partitioning [5]. Partitioning such an application with minimum distributed communication is important. The usage of coign, which automatically distributes applications conforming to Microsoft Corporation's Component Object Model (COM), is well discussed in Automatic Distribution Partition of Component Based Application which explains importance of automatically partitioning and distributing component based applications [6].

Several application partitioning problems haves been analyzed as an instance of the graph partitioning problem. Many heuristic based graph partitioning algorithms are used for partitioning purpose. Among those Kernighan-Lin heuristic based algorithm is heuristic method for partitioning arbitrary graphs which is both effective in finding optimal partitions and fast enough to be practical in solving large problems [7]. Similarly Min-Max cut greedy partitioning algorithm is also used to produce an initial bisection; it starts from two seeds and growing their own regions by adding candidate vertices in turn [8]. Fiduccia-Mattheyses (FM) heuristic for bi partitioning circuit hyper-graphs is an iterative improvement algorithm [9]. Well known parallelization of graph partitioning algorithm METIS is used for graph partitioning which has three main stages: (a) Coarsening takes a large graph, with vertices |V| and edges |E|, and creates successively smaller graphs that are good representations of the original graph; (b) Partitions the small graph and (c) Projects and refines the partition of the smaller graph onto the original graph [10]. Improvement factor over graph partitioning was introduced by Davidson algorithm [11].

3. Application Partitioning Approach for Massively Multiplayer Game.

Multiplayer game can be characterized based on entities and events. Entities and events can be specific to client, server or common to both. Examples of client entities are console, avatars, walls, weapons, etc. while examples of server entities are monster, war ship, tank, weapons, etc. Examples of client events are mainly graphics rendering, input handler, actions of avatars, shooting, and running, etc. while examples for server events are client update, connection management, game state update, coordinates computation and other mathematical functionalities. Similarly examples for common events are network connection management, sound, system utilities, display management, etc. A game, which follows client server model, has client logic and server logic sharing some common game logic. To partition such as application requires certain factors to be considered. Major factors to be considered for application partitioning are: (a) Availability of memory; (b) Processing speed on both client and server and; (c) Bandwidth for the amount of data that can be transferred over the network with minimum latency. Apart from above mentioned factors one need to consider others factors specific to application being partitioned such as: (a) Client and server tasks specific of a particular event; (b) Dependencies between the tasks across client server; and (c) Number functional of calls on tasks for a particular event.

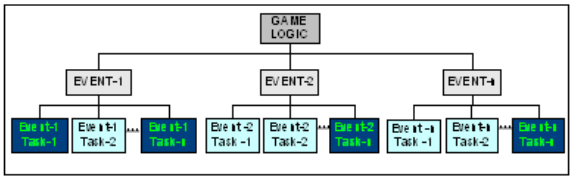

Figure 1. Model of Game Design

In our approach we model the game application, which follows client server model. We consider client/server processing speed, memory availability and set of game logic represented in the form of set of events as shown in figure 1. Each event comprises of certain tasks and each task having some game logic executed on client or server with dependencies between the tasks and functional calls between tasks across client server. The performance of game is measured in terms of total execution time. We compute the total execution time taken by each of the events under varying resource constraints on both client and server, assuming network bandwidth is good with negligible constant latency. To address the problem of partitioning we represent our application in terms of a weighted graph and partitioning problem reduces to graph partitioning problem. To perform graph partitioning, each of the tasks is represented as node and dependencies/functional calls between each of the task are represented as weighted edges. We assign edge weight considering the number of dependencies between the tasks. We use Kernighan-Lin Heuristic Based Algorithm for partitioning our application represented in graph [7].

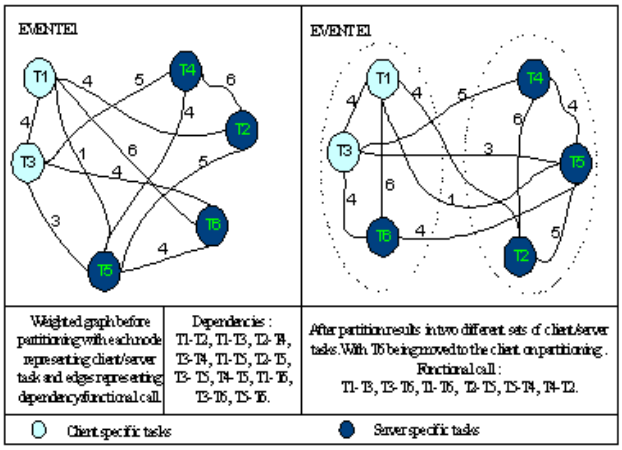

Figure 2. Modeling Game Design as Weighted Graph

This algorithm is an iterative algorithm, starting from a load balanced initial bisection. The initial bisection is generated based on the task dependencies. The algorithm first calculates, for each vertex, the gain in the reduction of edge-cut that may result if that vertex is moved from one partition of the graph to the other. It mainly has two iterations, inner and outer respectively. At each inner iteration, it moves the vertex, which is unlocked (vertices which are not swapped) having the highest gain, from the partition in surplus (that is, the partition with more vertices) to the partition in deficit. This vertex is then locked and the gains updated. The procedure is repeated even if the highest gain may be negative, until all of the vertices are locked. The last few moves that had negative gains are then undone and the bisection is reverted to the one with the smallest edge-cut so far in this iteration. This completes one outer iteration of the K-L algorithm and the iterative procedure is restarted. When an outer iteration fails to result in any reductions in the edge-cut or load imbalance, the algorithm is terminated. The K-L algorithm is a local optimization algorithm, with a limited capability for getting out of local minima by way of allowing moves with negative gain. Figure 2 shows weighted graph of our model before partitioning and after partitioning using KL algorithm. Figure 4 shows an instance of partitioning our game design. On partitioning the application is observed for each client's total execution time for each event. Thus obtained result is compared against the values obtained before partitioning.

4. Experimental Setup

We perform application partitioning on our modeled game design, having certain tasks on client and server. We set up a server with 5 tasks and with 20 client instances with each client having 2 tasks respectively. We construct weighted graph considering client and server tasks and number of dependencies/functional-calls as our attributes for graph. Application partitioning is achieved using KL-algorithm as described in section III. After partitioning certain tasks moved from server to client which results remote functional call affecting the network latency. We set up our application on two different machines on LAN with client having processing speed of 2.4 GHz and memory of 512Mb RAM and server with processing speed of 2.4 GHz and memory of 1 GB RAM. To vary the memory and CPU utilization, we run dummy applications to occupy memory and CPU utilization accordingly.

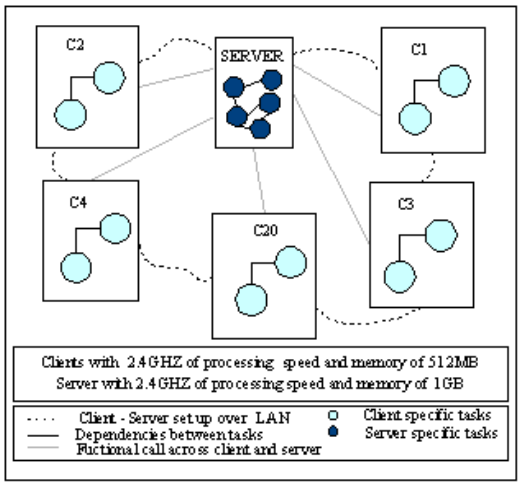

Figure 3. Experimental Setup of Game Model

We measure the total execution time taken by each of the client before and after partitioning under various conditions and infer the usefulness of our approach.

4.1 Results and Discussion

We perform application partitioning for typical four conditions as shown in table 1. We set up a server having 5 tasks and with 20 clients instance each having 2 tasks on two machines connected over LAN. To estimate the impact of partitioning, we perform the experiment by: (a) Moving one task from server to client; (b) Moving 2 tasks from server to client; and (c) Moving 3 tasks from server to client. We measure total execution time taken by each client before and after partitioning.

4.1.1 Scenarios:

4.1.1.1 Scenario 1: Both client and server resource utilization is low.

This is achieved since the only application running on these machines is the game application (with no other dummy applications utilizing the resources). Fig 5, 6 and 7 shows the difference in execution time taken by each client under aforementioned conditions. From the results obtained, we observe that there is a difference of 3.8% in total execution time after partitioning.

From the above observation, we conclude that application partitioning does not result in significant improvement over performance when both client-servers resources utilization is low.

4.1.2 Scenario 2: Both client and server resource utilization is high.

This is achieved by executing other dummy applications on both server and client machines along with the game application resulting in maximum resource utilization. Fig 8, 9 and 10 shows the difference in execution time taken by each client under aforementioned conditions. From the results obtained, we observe that there is a difference of 6.6% in total execution time after partitioning. From the above observation, we concluded that application partitioning is useful when both client and server have higher resources utilization leading to better performance.

4.1.3 Scenario 3: Server resource utilization is more than client.

This is achieved by executing other dummy applications on server machine along with the game application resulting in maximum resource utilization (with no other dummy applications utilizing client resources). Fig 11, 12 and 13 shows the difference in execution time taken by each client under aforementioned conditions. From the results obtained, we observe that there is a difference of 51.3% in total execution time after partitioning. From the above observation, we conclude that application partitioning results in significant improvement over performance, when server resource utilization is high.

4.1.4 Scenario 4: Client resource utilization is more than server.

This is achieved by executing other dummy applications on client machine along with the game application resulting in maximum resource utilization (with no other dummy applications utilizing server resources). Fig 14, 15 and 16 shows the difference in execution time taken by each client under aforementioned conditions. From the results obtained, we observe that there is an increase of 1.1% in total execution time after partitioning. From the above observation, we conclude that application partitioning on server is not useful, when client resources utilization is high.

5. Conclusion

In this paper we have addressed the scalability issues for gaming server on peak load conditions for massively multiplayer mobile gaming. Our approach is based on: (a) Modeling game application as graph; (b) Using graph partitioning to identify partition of gaming application for offloading; (c) Comparing application performance in terms of latency (before and after partitioning). We have instantiated our approach by modeling a gaming application and performing graph partitioning, using KL graph partitioning algorithm. Our experimental results have demonstrated that application partitioning is useful under: (a) Server resource utilization is more than client; (b) Both client and server resource utilization is high. Under following conditions application partitioning does not result in significant improvement on performance: (a) Both client and server resource utilization is low; (b) Client resource utilization is more than server. We plan to extend our approach by considering finer attributes of game design such as: Heterogeneity of client

resources; (b) Variability of network latency; (c) Latency of switching after partitioning etc.

6. REFERENCES

[1] C. Brown, P. Barnum, D. Costello, G. Ferguson, B. Hu, and M. Van Wie, "Quake II as a Robotic and Multi-Agent Platform," The University of Rochester, Computer Science Department, Rochester, New York 14627, Technical Report 853, Nov. 2004.

[2] G. Deen, M. Hammer, J. Bethencourt, I. Eiron, J. Thomas, and J. H. Kaufman, "Running Quake II on a grid," IBM Systems Journal, VOL 45, NO 1, 2006.

[3] B. Grüter, A. Mielke, and M. Oks, "Mobile Gaming - Experience Design," *3rd International Conference on Pervasive Computing – Pervasive*, Munich, Germany, 2005 - 8-13 May, 2005.

[4] "Multiplayer Mobile Games: Business Challenges and Opportunities," Forum Nokia, Version 1.0, April 16, 2004.

[5] M. Ball, C. Cifuentes, and D Bairagi, "Partitioning of Code for a Massively Parallel Machine," Sun Microsystems Laboratories, Menlo Park CA, Nov. 2004.

[6] G. C. Hunt, "Automatic Distributed Partitioning of Component-Based Applications," The University of Rochester, New York, 1998.

[7] W. B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell System Technical Journal, 1970.

[8] C. Ding, X. He, H. Zha, M. Gu, and H. Simon, "A min-max cut algorithm for graph partitioning and data clustering," *Proc. IEEE Int'l Conf. Data Mining*, 2001, pp.107-114.

[9] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," *19th Design Automation Conference*, 1982, pp. 175-181.

[10] Z. Kasheff, "Theory of Parallel Systems, Partial Parallelization of Graph Partitioning Algorithm METIS," Term project, Métis Graph partitioning software.

[11] M. Holzrichter and S. Oliveira, "A graph based Davidson algorithm for the graph partitioning problem," *International Journal of Foundations of Computer Science*, 1999.

Table 1. Application Partitioning Performed under Four Typical Conditions.

Scenario	CPU Usage		Memory Usage	
	Client	Server	Client	Server
Both client and server resources utilization are low.	10-50%	10-50%	20-50%	20-50%
Both client and server resources utilization are high.	95-100%	95-100%	70-90%	70-90%
Server resource utilization is more than client.	10-50%	95-100%	20-50%	70-90%
Client resource utilization is more than server.	95-100%	10-50%	70-90%	20-50%

Instance:

Instance of Algorithm in our game design for weighted graph as shown in fig 2. Consider event E1 with tasks {T1, T2 T3, T4, T5, T6}.
We construct weighted graph 'G' with G = (N, E, WE) to be partitioned as N = A U B, {|A| = |B|} Where, 'G' is the weighted graph. 'N' is the set of nodes represented as tasks T= {T1, T2, T3, T4, T5, T6}. 'E' is the set of edges and 'WE' represents edge weight between nodes by considering number of dependencies 'A' represents set of Client tasks and 'B' represents set of server tasks respectively in the game.

We take initial partitions to be:
Partition A: T1, T2, T3
Partition B: T4, T5, T6
Compute initial partition cost and 'D' for all tasks of partition 'A' and 'B'

Iteration 1:
Calculated value of gain(g) is:
g0(T1, T4) = 6, g1 (T1, T5)= -2, g2 (T1, T5) = -7, g3 (T1, T6)= 2, g4 (T2, T4) = - 2, g5 (T2, T4) = 13, g6 (T3, T4) = 5, g7 (T3, T5) = 3, g8 (T3, T6) = 6.
Maximum gain(g5) = 13
Swap (T2, T6).
Update D values for remaining nodes: A (T1, T3) and B (T4, T5).
=>A: T1, T6, T3
=>B: T4, T5, T2

Iteration 2:
Calculated value of gain for remaining nodes in A, B is:
g0(T1, T4) = -10, g1 (T1, T5) = -8, g2 (T3, T4) = -15, g3 (T3, T5) = -7
Maximum gain(g2) = -7.
Swap (T3, T5).
Update D values for remaining nodes in A (T1) and B (T4).
=>A: T1, T6, T5
=>B: T4, T3, T2

Iteration 3:
Calculate value of gain for remaining nodes in A, B is:
g0(T1, T4) = -6
Maximum gain(g0) = -6
Swap (T1, T4).
=>A: T4, T6, T5
=>B: T1, T3, T2

The set of partial sums of the max 'g' at each iteration is: 13, 6, and 0.
New partitions are: newA (T1, T3, T6) and newB (T2, T4, T5).

Exchange the first pair (T6, T2) from sets newA (T1, T3, T6) and newB (T2, T4, T5) to keep the partition cost minimum

Iteration 1:
Calculated value of gain(g) is:
g0(T1, T2) = -21, g1 (T1, T4) = -10, g2 (T1, T5) = -8, g3 (T3, T2) = -7, g4 (T3, T4) = -13, g5 (T3, T5) = -15, g6 (T6, T2) = -7, g7 (T6, T4) = -11, g8 (T6, T5) = -15.
Maximum gain(g2) = -7.
Swap (T3, T2).
Update D values for remaining nodes: newA (T1, T6) and newB (T4, T5).
=>newA: T1, T2, T6
=>newB: T3, T4, T5

Iteration 2:
Calculated value of gain for remaining nodes in newA, newB is:
g0(T1, T4) = -8, g1 (T1, T5) = -4, g2 (T6, T4) = -1, g3 (T6, T5) = -3
Maximum gain(g2) = -1.
Swap (T6, T4).
Update D values for remaining nodes in newA (T1) and newB (T5).
=>newA: T1, T2, T4
=>newB: T3, T6, T5

Iteration 3:
Calculate value of gain for remaining nodes in newA, newB is:
g0(T1, T5) = 8
Maximum gain(g0) = -8
Swap node (T1, T5).
=>newA: T1, T2, T4
=>newB: T3, T6, T5

The set of partial sums of the max 'g' at each iteration is: -7, -8 and 0.

Final set after partitioning are:
Partition A': T1, T3, T6.
Partition B': T2, T4, T5.

Figure 4. Instance of KL Algorithm for our Game Model

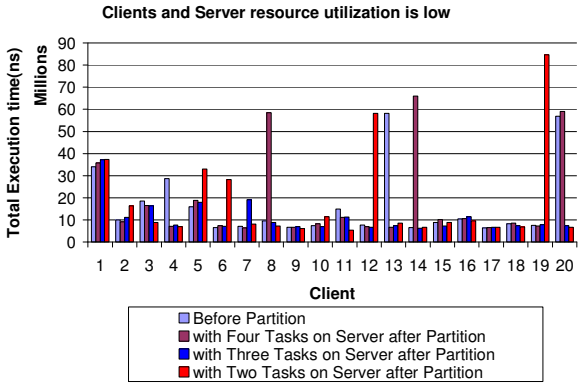

Figure 5. For Scenario 1

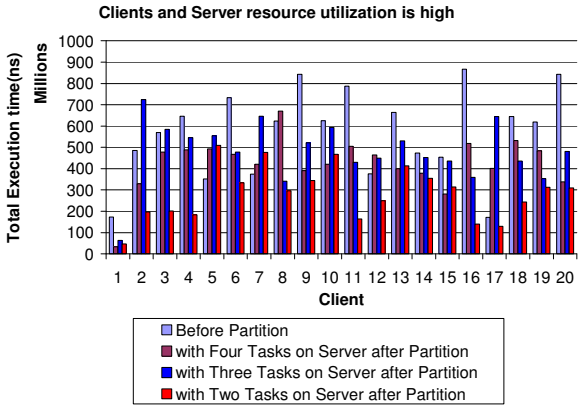

Figure 6. For Scenario 2

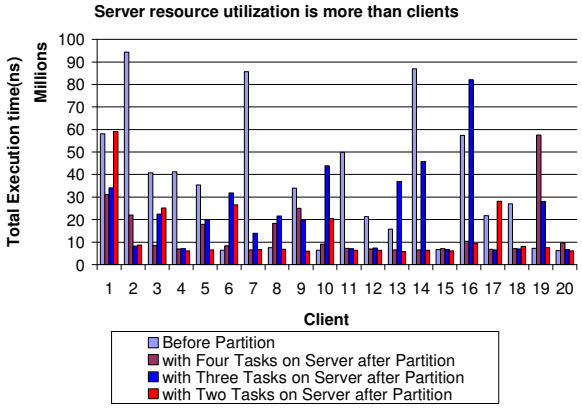

Figure 7. For Scenario 3

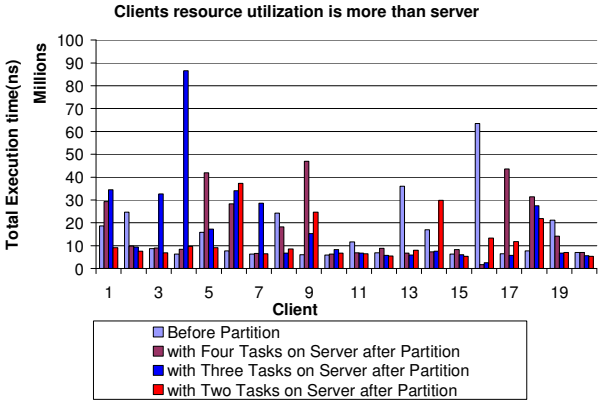

Figure 8. For Scenario 4