Context Aware Concurrent Execution Framework for Web Browsers

Aamir Saeed[1], Aiman Mahmood Erbad[2] and Rasmus Løvenstein Olsen[1]

[1]*Aalborg University, Denmark*
[2]*Qatar University, Qatar*
E-mail: {*asa; rlo*}*@es.aau.dk, aerbad@qu.edu.qa*

Received 28 January 2016;
Accepted 21 March 2016

Abstract

Computing hungry multimedia web applications need to efficiently utilize all the resources of a device. HTML5 web workers is a non-sharing concurrency platform that enables multimedia web application to utilize the available multi-core hardware. HTML5 web workers are implemented by major browser vendors to facilitate concurrent execution in web clients and enhance the quality of ambitious web applications. The concurrent execution in web workers allows parallel processing using available cores at the expense of communication overhead and extra computation. The benefits of concurrent execution can be maximized by balancing load across workers/CPU cores. This work presents load-balancing algorithms between web workers using parameters such as scheduler throughput, computation priority and game entities locality. An award-winning web-based multimedia game (raptjs.com) is used to evaluate the performance of load balancing algorithms. The preliminary results indicated that the performance of game improved with the proposed load-balancing across web workers. The load balancing algorithms were developed on top of DOHA [1], an open source JavaScript execution

This paper was made possible by NPRP grant #8-519-1-108 from the Qatar National Research Fund (a member of Qatar Foundation). The findings achieved herein are solely the responsibility of the authors.

Journal of NBICT, Vol. 1, 185–208.
doi: 10.13052/NBICT.2016.010

layer for multimedia applications. The load between web workers is transferred between web workers via serialized objects. Effects of load transfer between on the overall application performance was measured by analysing jitter and number of frames per second. Load balancing algorithms and load transfer mechanism can be used by developers to improve the application design, and giving end users better perceived quality of experience for demanding multimedia web based applications.

Keywords: Web workers, concurrent execution, load balancing.

1 Introduction

Performance has been improved steadily due to increased CPU clock speeds, and now the programming paradigms requires efforts to improve application performance using distribution and/or parallelism. One of the outcome of such an effort is HTML5 [2] framework or HTML5 web workers [3] that utilize the multi-core processors. HTML5 web workers provides an API to run scripts in background with out locking the user interface resulting in speedy execution of computational expensive applications.

The quality of ambitious application is also leveraged by HTML5 web workers concurrent execution. The concurrent execution of web workers is achieved by utilizing the available CPU cores to execute parallel tasks. Concurrency execution can be enhanced further by balancing loads between CPU cores. However transferring load or computation between web worker will always come at the cost of communication overhead and extra computation.

Games and other ambitious web applications are shifting the performance optimization focus from the download and parsing time of web files to the runtime performance. Memory, CPU cores and graphical capabilities of devices affects user experience positively or negatively of the web based games based on the resources specification. CPU cores are the units responsible for processing all the computation of the web based games. HTML5 web workers introduced a shared-nothing concurrency model as the first step toward concurrent web applications, as is shown in Figure 1. Although it provides concurrent execution but have limitations in terms of load balancing/sharing between cores at run time. Transferring or migrating computation across the CPU cores will leverage the concurrency performance. The load of the web based game on multi-core processors can be exchanged so as to maintain the load balanced across cores.

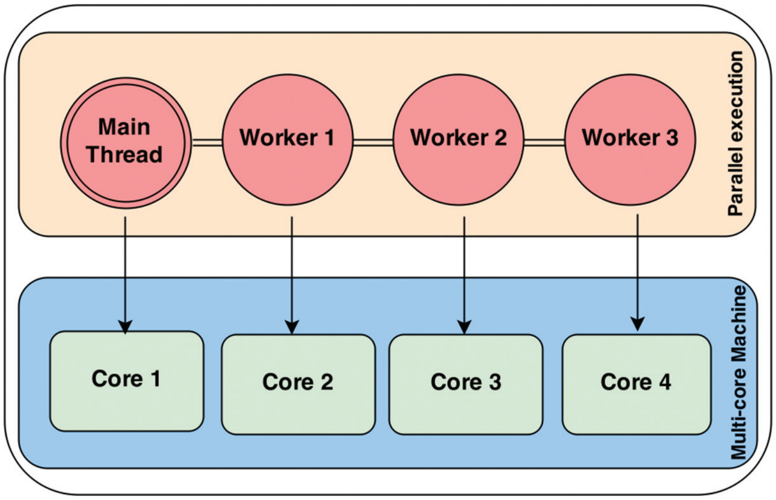

Figure 1 Multimedia application execution on a multi-core machine.

In this work a concurrent execution model is proposed that leverage the existing concurrent model that is hindered by overloaded core. The proposed approach exchange loads between CPU cores at run time based on either on the scheduler statistics or game world entities position with respect to game player position. In addition to load exchange mechanism, this work also presents load-balancing algorithms between web workers using parameters such as scheduler throughput, computation priority and game entity locality. An award-winning web-based multimedia game (raptjs.com) is used to evaluate the performance of load balancing algorithms. The preliminary results indicates that the performance of the game improved with effective load-balancing across web workers.

The proposed execution model and the three load balancing algorithms were developed on top of DOHA. It is an open source JavaScript execution layer for multimedia applications. DOHA provides publish-subscribe layer to support communication across concurrent execution threads (the main browser thread and worker threads). The load between web workers is transferred between web workers via serialized objects. The first algorithm uses the number of events dispatched by the event-loop (scheduler throughput) to balance load between cores. The web worker communicates the number of events dispatched per second to the main thread periodically. The main thread ensures the same number of events (i.e., computations) is processed in each worker and informs the overloaded worker to shed some of its load to less loaded workers (irrespective of their location). This algorithm has superior performance but it is only appropriate when most game entities have no locality constraints. For most games locality is critical, so the second and third algorithms respect game entities position in the game world while balancing loads. The second algorithm splits the game world into zones such that each zone has an equal number of entities. Each zone is assigned to a worker. As the game entities move in the world, the zones are updated periodically to maintain an equal load across workers. The third algorithm uses both locality and priority of entities to make load balancing decisions. Entity priority is defined by the contribution to the perceived quality using different spatial and temporal criteria, such as distance. This load balancing algorithm ensures that each web worker have the same number of high priority entities.

The performance of proposed algorithm was analysed by quantifying the delay incurred by each game entity due to processor execution of the simulation loop. A brief definition of jitter, frames per second (FPS) and priority is mentioned in evaluation section of paper, whereas a detailed description of it is mentioned in DOHA [1]. An award wining web based game RAPT is augmented with capability to exchange load between web workers at run time using serialization and de-serialization of game entities. The performance of the application showed improvement in performance by employing the load balancing algorithm. These techniques can be used by developers to improve the application design giving end users better perceived quality of experience for demanding multimedia web based applications.

2 Related Work

Concurrency available in modern hardware provides a framework for programming models to speed up application computations. Shared memory programming model was not accepted by web development community due to dead lock and live locks. Web development community adopted message passing programming models to achieve concurrency. For example HTML5 adopted web workers as a message passing programming model. However, message passing approaches in general suffers from high cost of communication.

Web development community exploit parallel hardware without changing their programming style and avoids to learn new language and learn new semantics. River Trail [4] provides development community a programming model that caters these needs of development community. It is a data-parallel programming model API using the concept of parallel array as a fundamental abstraction of parallel computation. The model allows spawned tasks immutable access to their parent's state and siblings are free to allocate and mutate their local heap. However, siblings cannot communicate with each other. River Trail uses a fine grained data-parallel model and on purpose avoids the concept of thread and scheduling. In contrast to this, our approach employs web worker concept to achieve parallelism. Each web worker scheduler process its events and communicates its state to main thread. Web workers communicate and exchange states using JSON Objects.

TigerQuoll [5] API provides JavaScript applications a framework to extends single-threaded event-based JavaScript programming with support of parallelism. At runtime it executes multiple event handler function in parallel to ensure consistent view of the shared memory. Meanwhile our approach used the event driven approach introduced in DOHA [1] for each web worker to executes its events. To keep a consistent view of shared states, the main thread is notified using a global event notifier. Our approach is more focused for multimedia based client side JavaScript application.

Although the traditional approach of loading a web page is mainly constraint due to the blocking manner of processing of script objects. In Fast and Parallel [6], bottlenecks due to parsed representation of page and its processing were analysed. It introduced algorithms to minimize the bottlenecks due to CSS selectors, layout and rendering objects in loading a web page. It applied parallelism within every step such as during CSS selectors matching, box and text layout, glyph handling, and painting and rendering.

ParaScript [7] focused on dynamic parallelism of compute-intensive JavaScript application. It performs automatic speculative parallelization of loops and generate multiple threads to concurrently execute iteration in these applications. It also introduced a novel speculation system that detects miss-speculations occurring during parallel execution and rolls back the browser state to a previously correct checkpoint.

In DOHA [1] adaptation policies were developed for game entities to define relative importance among different game entities in each game loop iteration. Priority was introduced to dictate events execution based on the game entities distance from the game players. It also implemented an execution layer on top of HTML5 web workers to enable scalable quality in interactive multimedia application. Concurrency issues such as state management and load balancing was dealt using event driven programming model. In our work we augmented the concurrency execution further by incorporating multimedia applications objects as transferable across web workers based on one of following heuristics i.e., event loop throughput (scheduler), proximity of objects in game world and priority of objects.

ParallelJS [8] proposed a framework, that enables developers to use high performance data parallel GPUs in standard web applications. Applications are written in combination of JavaScript and the constructs defined in ParallelJS library. The program can be either executed in either CPI or translated to PTX and executed in the GPU.

Existing approaches mentioned in this section and others available in web browser and literature tackles concurrent execution at a various levels. For instance utilizing parallelism for layout, modularizing the task and executing them, use of GPU for specific processes, and utilizing multi-cores architectures. In this paper, DOHA, an event driven programming model that focused on introducing parallelism and prioritization criteria to minimize the blockage of script objects is leveraged with runtime load sharing capability. The fine granularity defined in DOHA in terms of events type and its execution priorities provided a platform to identify each web worker workload. Based on workload identification events scheduler is dynamically altered to offload loads across web workers. This paper presents the core idea of load sharing across web workers using different load balancing algorithms for enhancement of concurrency execution of web browsers.

3 Design and Implementation

Concurrent execution of browser performs better than the single-threaded execution browser. DOHA, an execution layer on top of JavaScript engines that enhances event-driven concurrency model introduced a MultiProc API. This API provides a platform to develop concurrent web applications with centralized and distributed architecture based concurrency. In this paper a load sharing and load-balancing algorithms were proposed and tested on top of the DOHA concurrent execution engine. The platform partitions the game entities statically and distributed excessive loads by employing load balancing algorithms.

3.1 Load Balancing

Load balancing algorithm evenly distribute load between resources. Each web worker load is measured periodically and evenly balanced using the proposed load balancing algorithms. For measuring load of each web worker, the event loop throughput (scheduler throughput) defined in DOHA, position of application entities and the position entities in proximity of players in the game world were identified and used as load metrics. Moreover, a communication architecture provided in DOHA is adopted to exchange both load status messages and excessive load of web workers. All the communication of messages is exchanged between web workers via the main thread. The main thread is also empowered to make load balancing decisions. JavaScript based multimedia application uses main thread for the simulation of the game entities at each tick. The availability of all entities information at each tick make it ideal design choice to have a centralized decision making.

1) *Event loop (scheduler) throughput (Events dispatched per second):* Players movement in a game world changes computation load at web workers. Web worker scheduler throughput is one of the option to considered to quantify load. Heuristic based decision are appropriate for real time decision system. Exchange of heuristics between web workers and main thread is depicted in Figure 2.

Figure 3 shows the message sequence diagram for scheduler throughput based load balancing algorithm. Each web worker sends its respective scheduler throughput periodically to main thread. Upon reception of each worker throughput, the main thread computes average load and label worker

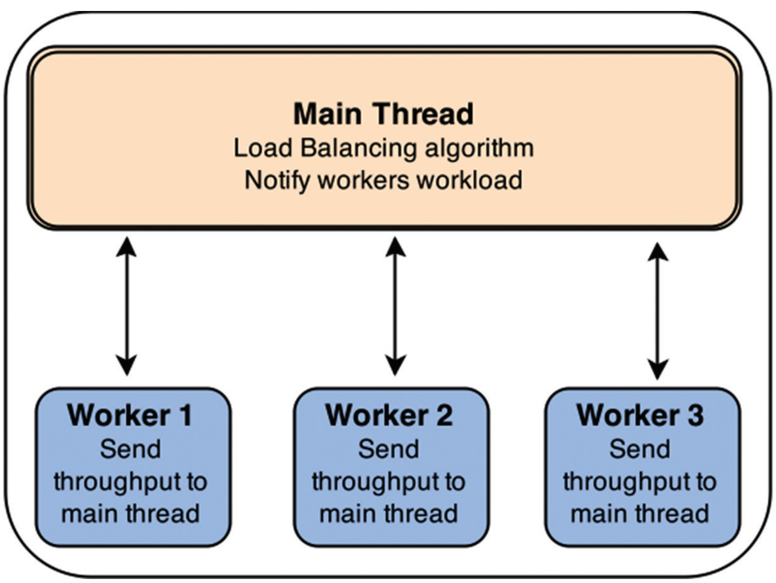

Figure 2 Load balancing using scheduler throughput.

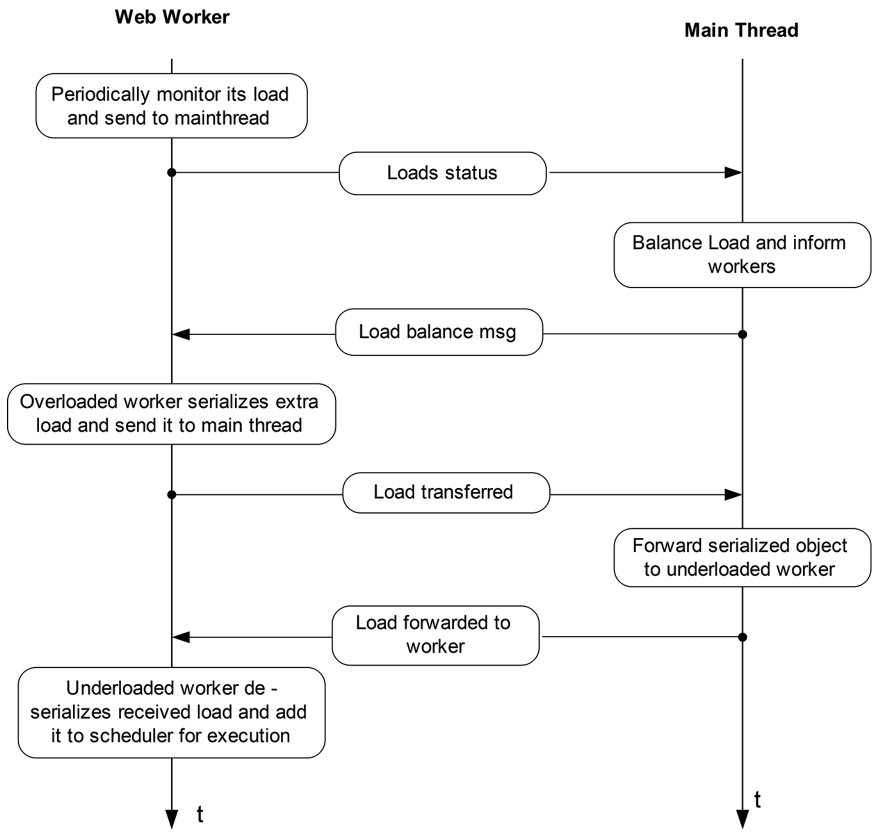

Figure 3 Sequence of events between worker and main thread.

as either overloaded or underloaded. The excessive load of web workers is distributed among underloaded web workers. The main thread direct the web workers to offload some of game entities to underloaded web workers based on percentage set by the main thread. The overloaded web worker offload the load to respective underloaded web worker and the underloaded web worker add the new workload to its scheduler.

Table 1 represents the information collected by main thread in order to make load balancing decision. In the table, wid represent web worker ID, l_k represent current load of a worker with ID k, μ as average load of all the worker, and Δ_k as difference between current load and average load. It shows an example of how the main thread evenly compute load between worker. In the table one can see that $wid1$ and $wid3$ are overloaded with 33.75% and

Table 1 Load balancing using events dispatched per second

wid	l_k	$\mu = \frac{1}{n} \sum_{k=1}^{n} l_k$	$\Delta_k = \lvert\mu - l_k\rvert$	$\frac{\Delta_k}{l_k} * 100$	Status
1	400	265	135	33.75%	Overload
2	200	265	65	32%	Under-load
3	180	265	85	47%	Under-load
4	280	265	15	5%	Overload

5% of extra load respectively. To avoid unnecessary migration of entities we introduced a criteria to consider a worker overloaded if its load exceed by 10% from the average. Only web worker i.e., $wid1$ is instructed by the main thread to offload its 33% of load to underloaded workers.

2) *Locality as load balancing:* Game world is a space and objects lies in the space. The objects manifest itself by moving, colliding and interacting with other objects. The locality or spatial information of an object at any instant of time is critical for the game engine and processor. Effectively utilizing locality information for concurrent execution of multimedia application can leverage the performance of computing. For instance assigning load to web workers based on locality for a game that have 10 players and 400 enemies and each web worker process only those player and game entities that lies in a space assigned to a web worker. Object movement such as of player and enemies are reported to another web worker if the object have moved to a space that is not in jurisdiction of the current web worker. These object need to be transferred as well to another web worker but is limited with design constraint of the architecture to exchange states with each other. On the contrary if these objects are not transferred to other web worker and movement of objects result in unfair load uneven distribution based on initial distribution. The locality as a metric for load balancing algorithm was proposed to overcome this scenario.

As a use case a load balancing scheme using locality is presented where each web worker load is assigned based on the spatial attribute of the objects in the game world. The game world is equally partitioned to each web worker based on the spatial attribute of object as shown in Figure 5. The entities in each game world partition can be processed by the respective web worker. The partitioning of game world can be either done using the vertical axis or horizontal axis or a combined one.

For vertical partition the game entities are sorted based on x-axis and equal number of entities are kept in each partitions and boundaries of partition are computed as shown in Figure 5. Figure 5 shows the game world where non-playing entities (enemies) in game world and players are seen. For executing the algorithm the entities along *x-coordinate* are obtained and all the entities *x-coordinate* values are sorted to define boundaries for partitions. The number of partition is equal to number of worker being employed. Each web worker event-loop is assigned all the entities for execution in one partition (rectangular region). Similarly horizontal based partition are computed based on *y-coordinate* of entities position. For two dimension approach, first entities are sorted based on either *y-coordinate* or *x-coordinate* and entities are assigned to partitions (first level). Then each partition can be further portioned (second level) based on alternate coordinate of the first one (if first level portioning was x-coordinate then second level will be y-coordinate based and so vice versa).

Locality based approach was introduced so as to avoid situation where a web worker for a web based game is not fully utilized. One of the web worker might be processing very few entities in comparison to another web worker which might be processing a huge number of entities. Periodically readjusting the web worker load based on locality effectively utilize idle web workers.

Web worker communicates the position of entities to the main thread in each game tick for graphical simulation. Web worker piggybacks additional information (*webworkerid, idofentities*) to the main thread (see Figure 4). In this case the web worker does not monitor load periodically but rather add extra information in its message to main thread. The main thread is now assigned additional responsibility of monitoring each worker load. Main thread execute the locality based algorithm periodically to compute entities assignment to web workers. Each worker is notified about the entities and their respective ids of web worker to whom these entities are to be migrated. Entities states are shared between workers using serialized objects. Figure 4 deficit the events sequence diagram for the locality based load balancing algorithm.

The main thread periodically computes entities assignment to web workers based on game entities positions along the horizontal axis in the game world. The identification number of the game entities and its newly assigned web worker is maintained in a data structure and sent to the web worker to which these entities belong. Efficient data structure such as HashMap are used for computation of entities assignment to web workers by sorting all the entities in list and assigning them equally to web worker in limited interval of time i.e a game tick (for loops based sorting algorithm were affecting the performance). The main thread transfer the data structure consisting of entity ids and workers ids to the web worker to which these entities belong.

The web worker upon reception of the data structure, serializes the entities and sends the list of entities to the main thread. The main thread sends serialized

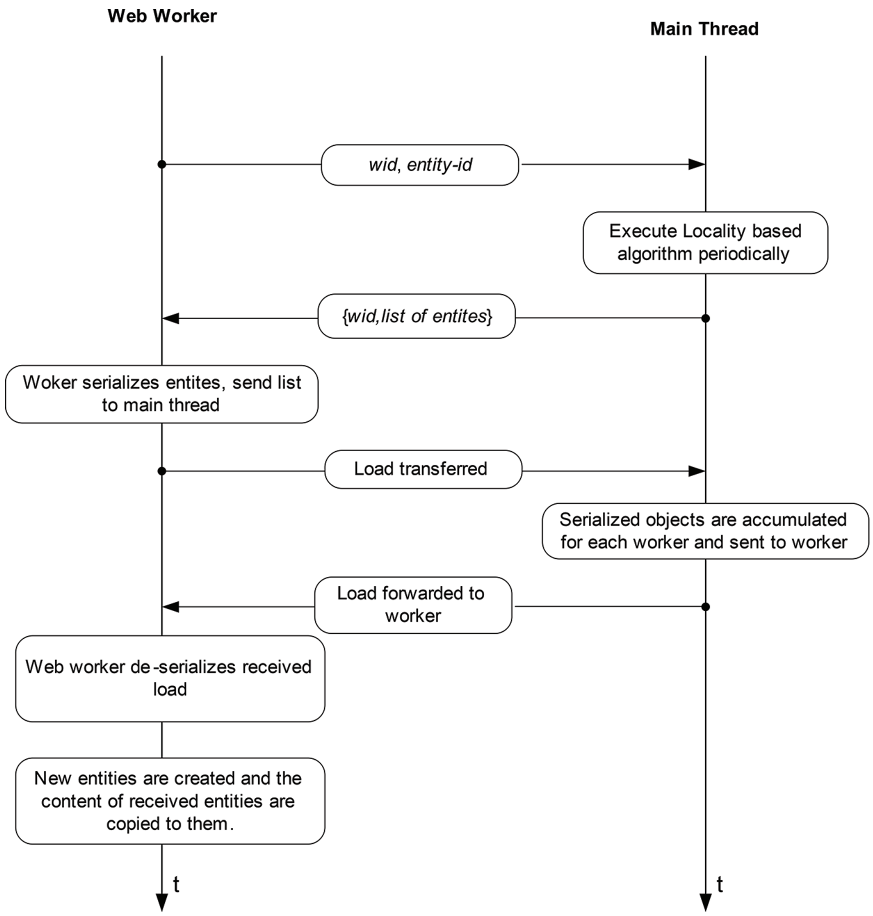

Figure 4 Locality algorithm sequence diagram.

entities to web worker based on the receiver worker id field specified in data structure. The web worker upon receiving the entities de-serializes the entities and create new entities and copy the content of the de-serialized object into new entities.

3) *Priority of entities as load balancing:* Objects in a game world can have different level of affect on gaming. Players may belong to different categories of membership i.e., gold member, silver member, non-paying members or players can have different level of earned points. Likewise enemies can also have different categories i.e., causing destruction by more than one degree of freedom or causing destruction by one mode such as throwing fire balls or

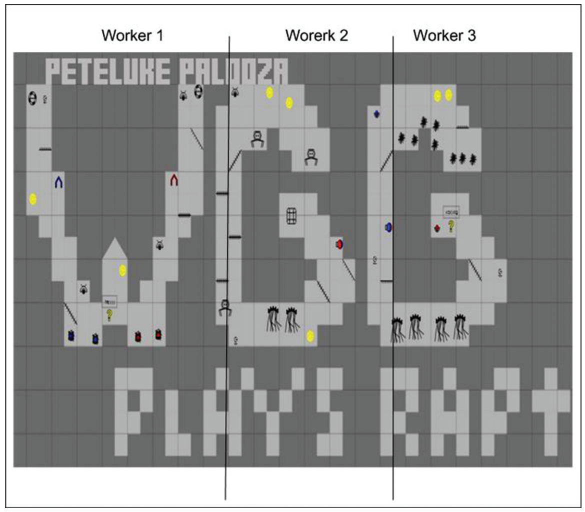

Figure 5 Vertical partitions of game world for 3 web workers.

causing death of player by contact etc. These special attribute of object in a game can be mapped into priority.

As a use case enemies distance from a player was selected as a priority to distribute load between web workers. The entities close to the player have higher priority than distant entities because they are in the user view and their update rate define user perceived quality. Load is balanced by distributing highly prioritized entities to all the running workers equally.

The aim of introducing priority as a parameter to balance load among entities is that each worker executes high prioritized entities equally. The algorithm follow a similar communication mechanism as the locality based approach i.e., exchange between web workers takes place via main thread. The main thread periodically computes highly prioritized entities assignment to web workers. The priority of the game entities are piggybacked on the updates that the web worker publishes for the game tick simulation using publish subscribe.

4 Load Exchange Across Web Workers

One of the main contribution of this paper was the exchange of computation between web workers at run time using the three proposed algorithm. HTML5 concurrency model main thread distributes work among the web worker at the start up. The web worker executes the computation using the available resources and communications between main thread and web worker takes place via the HTML5 message passing approach. In this paper a novel mechanism of offloading computation from a web worker at run time to another web worker using serialization and de-serialization technique is proposed to balance load between web workers.

To achieve the aim of offloading computation between web workers at run time JSON methods of JavaScript API were investigated. However, JSON methods i.e, *stringify* and *parse* used for serializing and de-serializing object have limitation when objects have circular dependency.

$$JSON.stringify(entity)\ JSON.parse(data)$$

Due to limitation of JSON methods in terms of circular dependency, the CircularJSON [9] API was used to serialize and de-serialize the objects which have circular dependency.

$$\begin{aligned} &var \quad serialized = CircularJSON.stringify(object) \\ &var \quad deserialized = CircularJSON.parse(serialized) \end{aligned}$$

The entities at source web worker is serialized using CircularJSON *stringify* method. The entity is removed from scheduler of current web worker and added to a list of transferable entities. Remote event is invoked to transfer the transferable list of entities to the main thread. The main thread forward entities to specified web worker based on load balancing algorithm. When a web worker receives an entity it de-serializes it and cloning function is invoked to create new entities. A deep copy function is initiated to copy the object content to the newly created entity. After cloning the entity, it is then added to this new web worker event loop (scheduler).

$$entityentity\_clone(newobject, deserializedobject)$$

5 Criteria for Entity Selection

Entities that needed to be migrated from one worker to another worker had to be selected through a fair mechanism. If fairness among the entities is not ensured then their is a possibility that some entities had to migrate between web workers more frequently. The load balancing algorithm employed picks objects for migration with out following any specific criteria and their is possibility that some entities can be picked again for migration which would affect the simulation of such entities. All the entities have uniform probability and a random entity can be picked at a worker for migration. Let $F(x)$ denote CDF and $x \in \mathbb{R}$ and $F : \mathbb{R} \to [0, 1]$.

$$F(x) = P(X \leq x)$$

and its Inverse CDF is

$$x = F^{-1}(F(x)) \tag{1}$$

In order to modify the random entities selection criteria so that entities are not picked again for migration. A method was defined to construct the probability mass function (PMF) for each worker. The new method of constructing PMF is based on the wait time of entities at worker before being migrated. Equation 2 was introduced as procedure to obtain individual probability of an entity of being picked.

$$Pr(entity) = \frac{1}{1 + e^{-(t_i - t_o)}} * S \tag{2}$$

where $(t_i - t_o)$ represent the age (wait time).

Equation 3 gives a procedure to construct the probability mass function of entities at the worker. Where S_k is the normalization factor for an k^{th} entity. Equation 1 can then be used to select an entity for migration.

$$\sum Pr(entity_k) = S_k \sum^{N} \left[\frac{1}{1 + e^{-(t_i - t_o)}} - \frac{1}{2} \right] .2 = 1 \tag{3}$$

$$\begin{aligned} &\text{if } t_i \to \infty \text{ then} \\ &S_k \sum^{N} \left[\tfrac{1}{(1+0)} - \tfrac{1}{2} \right] .2 = 1 \\ &S.N = 1 \Rightarrow S = \tfrac{1}{N} \end{aligned}$$

When $S = \frac{1}{N}$ is a special case of probability of being picked when all the entity have a uniform distribution. It means each entity will have an equal probability of being picked.

On the contrary, at any instant of time t each entity will have different age and each will have different probability of being picked for migration. The entities which are newest at the web worker will have lowest probability of being picked up as compared to the ones which are oldest as shown in Figure 6. The figure shows 6 entities and each have a different age. Older entity have a more larger probability of being selected than the younger ones.

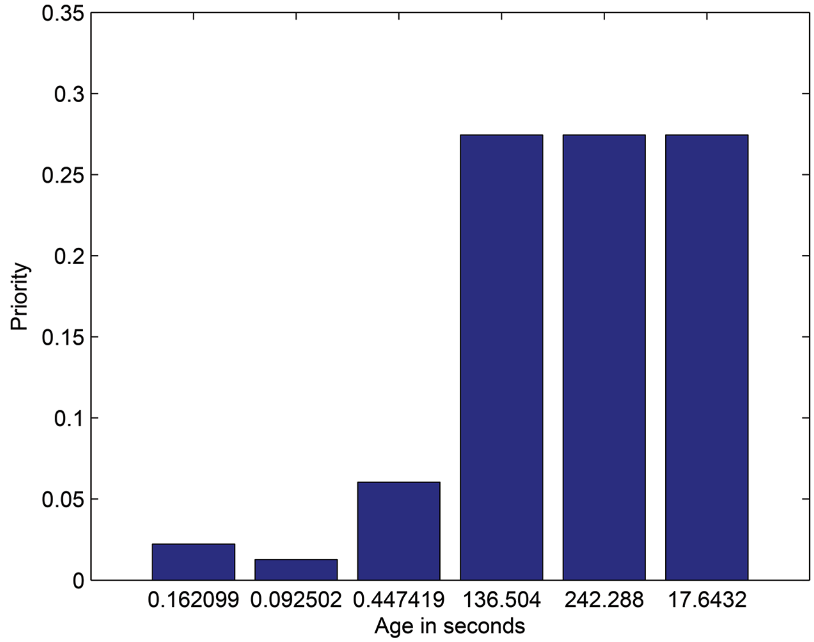

Figure 6 Entity selection based on sigmoid function.

Selecting entities based on age enable entities with higher probability to be selected more often for migration using all the algorithm mentioned for load balancing.

6 Evaluation

To evaluate the performance of the proposed algorithms, the quality of the game play experience using simulation loop jitters profile was analysed using average timeliness and mean frames per second (FPS) for all entities to quantify the average game quality. An overview of the performance evaluation metrics are presented as follows.

- **Priority:** Multimedia application game entities are executed within each loop iteration. In Doha [1], priority was introduced as a relative factor among game entities to schedule order of events execution. Players are the main actor in a game, and their executions and updates are most critical performance indicator of perceived game experience. All the entities in the game play are assigned execution priority based on its distance from an active player. Priority was defined as real value between 0.0 and 1.0. Players themselves are also entity and always has a highest priority. The priority assigned to the other entities updates events is inversely proportional to their distance from the closest player. Figures 7–9 show the performance of entities whose execution are scheduled based on its distance from player i.e., in terms of priority.

- **Jitter:** Jitter is the difference from the expected tick time. In 30 FPS game, each entity is expected to have a tick every 33 millisecond. If tick happened after 40 millisecond, then we have jitter of 7 millisecond. More jitter means the entity is not meeting its deadlines. In general, it is expected that high priority entities to have less jitter and more FPS.

$$Jitter = \lambda - (t_n - t_{n-1}) \tag{4}$$

where

λ denotes expected inter-arrival time (33 millisecond in 30 FPS)
t_{n-1} denotes time for previous tick
t_n denotes time for recent tick

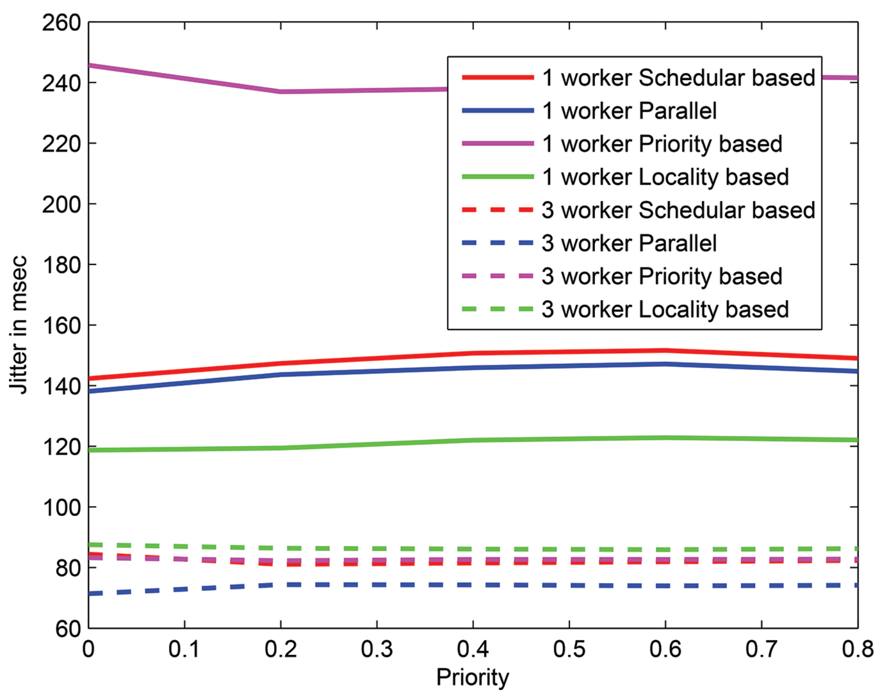

Figure 7 Jitter.

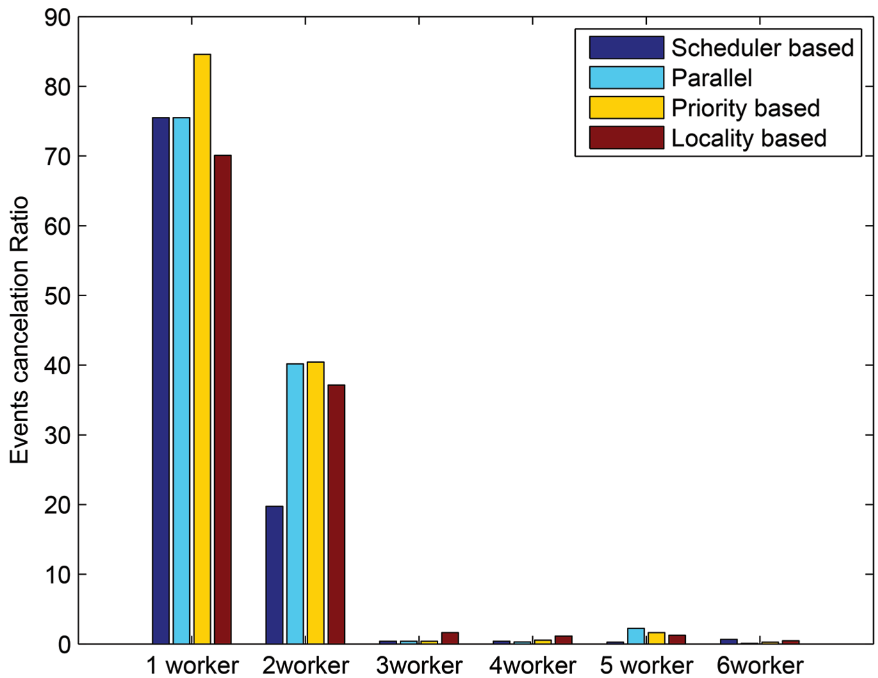

Figure 8 Algorithm comparison based on events cancellation ratio for parallel execution.

- **Frames per second (FPS):** The frames per second is the number of iterations of the simulation loop. In a typical multimedia application, we have 30 Frames Per Second (FPS) which means that we refresh the screen with updated content 30 times in each second (every 33.3 millisecond). This rate goes up to 60 frames per second. This concept is common in different multimedia applications, such as video, animation, and games. In traditional RAPT, all game entities updates are executed in each frame. In RAPT, the display loop frames which kept running at 30 frames per second is separated from the simulation loop frames (which update the state of game objects). So high priority entities will get executed (scheduled) in each cycle while less important entities (e.g., far from players) will be updated less frequently (cancelled if not executed by next tick). The average frames per second is an indicator which averages how many ticks on average each entity received per second (over all entities high priority and low priority). FPS is just measurement of how many updates happened for each entity in a second and taking the average over all entities.

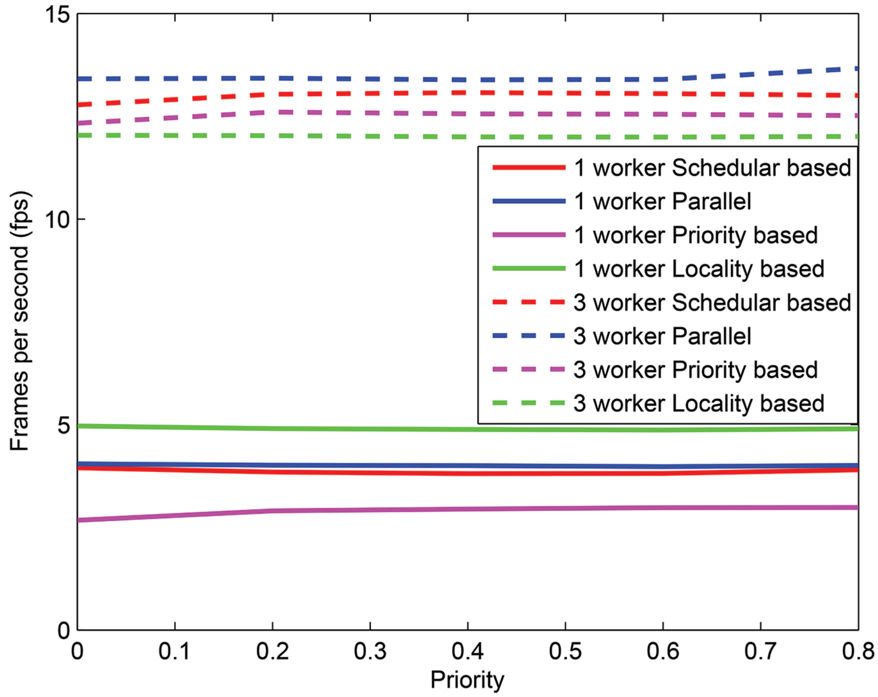

Figure 9 Algorithm comparison based on FPS for parallel execution.

- **Events Cancellation ratio:** Event Cancellation ratio is a metric to show the throughput of the event loop. The cancellation ratio increases when the frame duration is not enough to update all entities. Lower events cancellation ratio indicates better performance and higher cancellation ratio indicates performance degradation.

A set of experiment were carried out on a PC with following specifications i.e., Ubuntu 14.04.1 LTS, Google Chrome Version 39.0.2171.95 (64-bit), Processor: Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz (8 Cores) and RAM of 16 GB.

The performance of three proposed algorithm was compared with DOHA parallel execution model. A tough playing (hard) scenario of game was used so that the workers and main thread have huge amount load in order to evaluate the performance of the algorithm. The jitter profile is worst for one worker scenario as game loop iterate over all game entities in each frame leading to a large delay in processing each frame. With increase in workers the jitter improves from 250 msec to approximately 80 msec for all of load balancing algorithm as shown in Figure 7.

Event Cancellation ratio is a metric to show the throughput of the event-loop. Figure 8 show the event cancellation ratio for the algorithm. In case of 1 worker the event loop throughput is worst because the frame duration is not enough to update all entities for the hard scenario of game (more entities and high difficult level of game). An increase in the number of workers clearly shows improvement in event cancellation ratio. The Scheduler based load balancing algorithm exhibit better performance in terms of event cancellation ratio because the workers load were balanced based on the scheduler throughput. In case of two worker the scheduler based load balancing algorithm have better event cancellation ratio (least) as the frame duration is much better utilized to process all entities than the other scenario. As the number of workers are increased such as three, four, five and six, the event cancellation reaches lower end. Because now the events are not cancelled and all the algorithms have similar performance in terms of event cancellation ratio for number of workers more than two.

In Figure 9 improvement is seen for frames per second when the number of worker is increased from one worker to three workers. As the workers are increased further to six workers as shown in Figure 10 concurrent execution don't show further improvement. The reason is due to excessive communication between main thread and web workers. The three load balancing based concurrent execution algorithms has a higher overhead of communication than DOHA [1] concurrent execution algorithm. The periodic balancing of load and load exchange between web worker increases the communication overhead.

7 Conclusion and Future Work

Three load balancing concurrent execution approaches were implemented and tested to achieve better game quality. This work presents load-balancing algorithms between web workers using parameters such as scheduler throughput, computation priority and game entity locality. Preliminary results indicated that the performance of game improved with effective load balancing across web workers. Although periodic load balancing decisions for higher number of web workers added extra communication overhead which resulted in degradation of (FPS). Employing reactive load balancing techniques and modifying the communication pattern between workers and main thread are potential for improvement of the concurrency performance. In future work it is aimed to employ a reactive load balancing technique and up-gradation

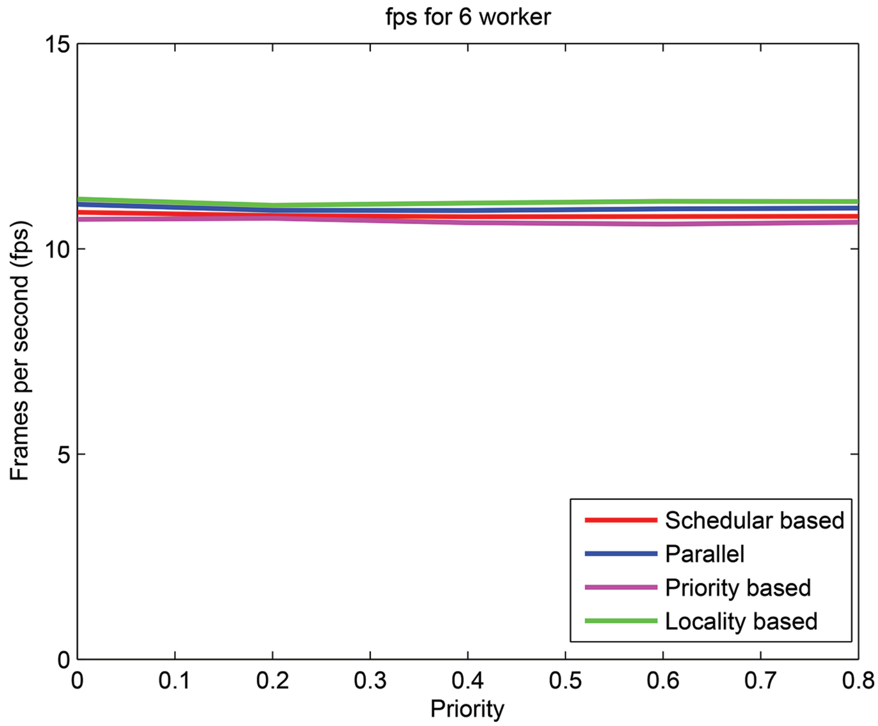

Figure 10 Algorithm comparison based on FPS for parallel execution.

of communication pattern to improve the performance of the concurrent execution of web browser execution.

References

[1] Erbad, A., Hutchinson, N. C., and Krasic, C. (2012). "Doha: Scalable real-time web applications through adaptive concurrent execution," in *Proceedings of the 21st International Conference on World Wide Web, WWW '12*, (New York, NY: ACM Press), 161–170.

[2] Berjon, R., Faulkner, S., Leithead, T., Pfeiffer, S., O'Connor, E., and Navara, E. D. (2014). *"HTML5," Candidate Recommendation, W3C*. Available at: http://www.w3.org/TR/2014/CR-html5-20140731

[3] Hickson, I. "*Web Workers*." Available at: http://dev.w3.org/html5/workers

[4] Herhut, S., Hudson, R. L., Shpeisman, T., and Sreeram, J. (2013). "River trail: a path to parallelism in javascript," in *Proceedings of the 2013 ACM*

SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA '13, (New York, NY: ACM Press), 729–744.
[5] Bonetta, D., Binder, W., and Pautasso, C. (2013). Tigerquoll: parallel event-based javascript. *Sigplan Not*. 48, 251–260.
[6] Meyerovich L. A., and Bodik, R. (2010). "Fast and parallel webpage layout," in *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, (New York, NY: ACM Press), 711–720.
[7] Mehrara, M., Hsu, P.-C., Samadi, M., and Mahlke, S. (2011). "Dynamic paral-lelization of javascript applications using an ultra-lightweight speculation mechanism," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture, HPCA '11*, (Washington, DC: IEEE Computer Society), 87–98.
[8] Wang, J., Rubin, N., and Yalamanchili, S. (2014). "Paralleljs: an execution framework for javascript on heterogeneous systems," in *Proceedings of Workshop on General Purpose Processing Using GPUs*, 72, (New York, NY: ACM Press).
[9] Giammarchi, A. (2009). *Circularjson.* Available at: https://github.com/WebReflection/circular-json.git

Biographies

A. Saeed is working as Postdoc at Aalborg University in the Wireless communication Networks (WCN). Aamir received his Ph.D. degree from Aalborg University in 2015 on the topic of Service Migration in Context Aware Environment with focus on Enhancement of User Quality of Experience and had his master and bachelor from Comsats Institute of Information Technology. His research interest includes transport protocols for internet, cloud services migration and Internet of Things (IoT).

A. Erbad is an Assistant Professor and Computer Engineering Program Coordinator at Qatar University. Dr. Erbad obtained a Ph.D. in Computer Science from the University of British Columbia, a Master of Computer Science in Embedded Systems and Robotics from the University of Essex and a Bachelor of Science in Computer Engineering from the University of Washington. His research interests span cloud computing, distributed systems and multimedia networking and systems.

R. L. Olsen is an Associate Professor at Aalborg University working in the Wireless Communication Networks (WCN) group. Rasmus received his master degree from Aalborg University in 2003 and has received his Ph.D. degree on the topic of Context Sensitive Service Discovery and Context Management with focus on access to dynamic information in 2008. Since, Rasmus have been teaching, supervising from 1st year master students to Ph.D. level, and working in various European projects and have more than 40 publications in papers for international conferences, journals and book chapters. Rasmus' current research focus is on the role of communication networks in smart grid and the impact of networks on remote access to dynamic data of the collection as well as the distribution of information. The key word is information quality metrics and quality of service in general information management systems.

Rasmus is also a member of the technical program committee of e.g., IEEE Smartgridcomm, IEEE VTC and IEEE NCA conferences, and is currently active in the electrical association of Intelligent Energy in Denmark. Previous Rasmus has previous been guest visitor at National Institute of Communication Technology in Yokosuka Research Park in Japan, cooperating with a team of researchers on next generation network technology. He is currently engaged in national and European research projects, currently MOBINET and EDGE, doing research and leader of work packages and was previously working as work package leader in SmartC2Net on Experimental Prototypes. Rasmus is the technical coordinator of the ForskEL (Energinet.dk) project RemoteGRID.