

Proyecto de Trabajo Profesional

Título

Utilización de esqueletos algorítmicos en EcmaScript para paralelizar el computo en navegadores Web a través de Web Workers

Using algorithmic skeletons in EcmaScript to parallelize computation in browsers through Web Workers

Autor/es

Schenkelman, Damián. Padrón 90728.

Servetto, Matías. Padrón 91363.

Tutor/es

Lic. Wachenhauzer, Rosa

Objetivo

En el año 1965 Gordon Moore propuso [1] la comúnmente conocida como "Ley de Moore" la cual predice que el número de transistores en un circuito integrado se duplicara cada dos años. Como consecuencia, los programas que utilicen un solo procesador/núcleo se volverán más rápidos sin necesidad de realizar modificaciones específicas para ello. En el año 2010 se predijo [2] que esta tendencia estaría llegando a su fin lentamente, principalmente por razones relacionadas con la disipación de calor de estos circuitos. Esto hace que las computadoras de última generación comiencen a tener una mayor cantidad de procesadores, en lugar de procesadores con más poder de computo.

Para poder aprovechar al máximo el rendimiento de estas computadoras, es fundamental que los profesionales de sistemas puedan crear programas que procesen datos en paralelo, mediante mecanismos como pueden ser hilos (*threads*) o procesos que se ejecutan en simultaneo, cada uno en diferentes procesadores.

En el caso de aplicaciones que se ejecutan en un único dispositivos cliente (ej.: navegadores y las páginas de internet, aplicaciones para dispositivos móviles, aplicaciones de escritorio), muchos lenguajes y plataformas proveen facilidades a los programadores para abstraer la complejidad de coordinación requerida por este tipo de procesamiento. Por ejemplo, la plataforma .NET provee las bibliotecas Task Parallel Library (TPL) y Parallel LINQ (PLINQ) [3] los cuales utilizan esqueletos algorítmicos con funciones de orden superior para que los desarrolladores puedan realizar operaciones complejas mediante una *API* simple.

Una de las tendencias que está tomando forma, es la utilización del lenguaje EcmaScript [4] (también conocido como JavaScript) para crear aplicaciones que normalmente eran solo consideradas en un ámbito nativo, por ejemplo los video juegos. Esto se ha logrado gracias a componentes como asm.js [5] y WebGL [6] en conjunto con el constante desarrollo de los navegadores y motores de ejecución del lenguaje EcmaScript.

En este contexto, uno de los planes del comité que desarrolla el lenguaje es proveer facilidades de este estilo para la versión 7 del mismo (ES7) con el fin de mantener la competitividad con aplicaciones nativas. Iniciativas como ParallelJS [7] o la posibilidad de ejecutar instrucciones SIMD [8] (*single instruction multiple data*) son algunas de las opciones para lograrlo.

El objetivo de este trabajo es proveer una alternativa para la ejecución de código en paralelo para EcmaScript, mediante una biblioteca que expondrá funciones de orden superior para modelar la implementación de esqueletos algorítmicos como *map*, *filter* y *reduce*. Esta biblioteca utilizara Web Workers [9] (que son el mecanismo de paralelismo propuesto por los navegadores hoy en día) para permitir la ejecución de código en simultaneo en diferentes procesadores. El proyecto será desarrollado de manera Open Source (sin aceptar contribuciones hasta el momento de su aprobación) utilizando un repositorio en Github <https://github.com>.

Alcance

Requerimientos Funcionales

En esta sección incluiremos los requerimientos funcionales de la biblioteca así como la documentación que detallará su funcionamiento y la investigación realizada para su desarrollo.

Notación y aclaraciones

Typed Array con elementos de tipo *A*: $TA\langle A \rangle$

Los tipos posibles de *A* al momento de escribir este documento son:

- Int8: entero 8 bits, entre -128 y 127.
- Uint8: entero 8 bits, entre 0 y 255.
- Uint8Clamped: entero 8 bits, entre 0 y 255. Las operaciones utilizan *Clamping* [10] en vez de aritmética modular en caso de excederse del rango permitido.
- Int16: entero 16 bits, entre -32768 y 32767.
- Uint16: entero 16 bits, entre 0 y 65535.
- Int32: entero 32 bits, entre - 2147483648 y 2147483647.
- Uint32: entero 32 bits, entre 0 y 4294967295.
- Float32: punto flotante de 32 bits siguiendo IEEE 754 [11].
- Float64: punto flotante de 64 bits siguiendo IEEE 754 [11].

Biblioteca

- Permitirá trabajar sobre Typed Arrays [12].

- Se evaluará la posibilidad de trabajar con *arrays* comunes del lenguaje [13]. Esto se debe a los altos costos de performance en los que se podría incurrir al serializar objetos comunes entre el *thread* de la interfaz de usuario (UI) y los *Web Workers*. En caso de ser posible serializar los datos sin incurrir en un alto costo de performance [14] se permitirá trabajar sobre *arrays* comunes.
- Tendrá una forma fácil de realizar la operación *map*.
 - La misma permitirá especificar una instancia de *Typed Array* de tipo *A* y una función *f* a ejecutar y opcionalmente un tipo de *Typed Array* de tipo *B* destino (si ninguno es especificado se asumirá $A == B$ para $TA<A>$ y TA).
 - La función *f* debe recibir un único elemento del $TA<A>$ y devolver otro elemento que pueda ser incluido en un TA.
 - La función *map* devuelve un TA cuyos elementos son el resultado de aplicar *f* a cada elemento de $TA<A>$.
- Tendrá una forma fácil de realizar la operación *filter*.
 - La misma permitirá especificar una instancia de *Typed Array* de tipo *A* y una función *f* a ejecutar.
 - La función *f* debe recibir un único elemento del $TA<A>$ y devolver *true* o *false* (o valores que representen esto en EcmaScript (por ejemplo *0* se considera *false*)).
 - La función *filter* devuelve un $TA<A>$ cuyos elementos son aquellos elementos del $TA<A>$ inicial tal que $f(a) === true$.
- Tendrá una forma fácil de realizar la operación *reduce*.
 - La misma permitirá especificar una instancia de *Typed Array* de tipo *A*, una función *f* a ejecutar y un valor inicial *v* de tipo *V*.
 - La función *f* debe recibir un único elemento del $TA<A>$ y el valor actual. El valor actual es *v* para el primer elemento de $TA<A>$ y el valor devuelto por *f* al procesar el elemento anterior para los elementos subsiguientes. La función *f* se llama para cada elemento de $TA<A>$ desde el primero al último.
 - La función *reduce* devuelve un *V* que es el resultado de la última llamada a *f*.
- Procesará todas las operaciones (*map*, *reduce* y *filter*) de forma asincrónica. Esto se debe a que la naturaleza del ambiente de ejecución no permite realizar operaciones sincrónicas al comunicar al *thread* de la UI con los *Web Workers*. Esta comunicación se realiza mediante pasaje de mensajes.
- Proveerá una forma de obtener los resultados de todas las operaciones de forma asincrónica. Alternativas posibles son *callbacks* o *promises* [15]. La decisión será tomada durante el desarrollo teniendo en cuenta factores como la performance, como afecta al tamaño (en bytes) de la biblioteca, facilidad de uso, entre otras.
- Las operaciones *map*, *reduce* y *filter* se ejecutarán de a una por vez. Debido a su naturaleza asincrónica, si una llamada a cualquiera de ellas se realizara mientras otra operación se encuentre en progreso, la misma se encolara. Así mismo, las operaciones se ejecutarán en el orden en que fueron iniciadas.

- Todas las funciones pasadas como parámetro a *map*, *reduce* y *filter* se serializarán para poder ser ejecutadas en los *Web Workers*, por estas razones no podrá accederse a elementos a través de su *closure*.
- Se podrá proveer objetos adicionales (incluye funciones) para cada una de las operaciones para suplantar la carencia de *closures*. El análisis tendrá en cuenta el tiempo de transferencia de los objetos y las características de los mismos. Por ejemplo, una opción sería permitir solo objetos completamente inmutables (es decir el objeto en sí y todas sus propiedades son inmutables [16] de forma recursiva).
- Se permitirá de alguna forma poder combinar las operaciones *map*, *filter* y *reduce*, permitiendo tomar ventaja de la propiedad de composición de las mismas. Un ejemplo en pseudocódigo:

```
var xs // array
var f, g // funciones
function compose(f, g){
  return function (x){
    g(f(x));
  };
}
xs.map(f).map(g) == xs.map(compose(f, g));
```

Adicionalmente:

```
var xs // array
var f, g // funciones

function compose(f, g){
  return function (x){
    g(f(x));
  };
}

xs.map(f).filter(g) == xs.filter(compose(f, g));
```

De esta manera, si se permite combinar estas operaciones en un único esqueleto a ser computado, se puede evitar el costo innecesario de serializar los datos entre los *Web Workers* y el *thread* principal. Una posible implementación (a modo de ejemplo, no necesariamente la que será utilizada) podría ser:

JavaScript

```
var xs // typed array
var f, g // funciones

var pjs = require('pjs');
var promise = pjs(xs).map(f).filter(g).seq();
promise.then(function(ys){
  // trabajar con resultado similar a ys == xs.map(f).filter(g)
});
```

- Se soportará alguna de las siguientes APIs de módulos de EcmaScript: AMD [17], CommonJS [18] o EcmaScript 6 [19]. La definición se realizará durante el transcurso del proyecto teniendo en cuenta diferentes factores.

Documentación

- Se escribirá documentación documentando cada una de las funciones públicas (de la API) de la biblioteca, incluyendo sus parámetros y restricciones, valores de retorno, condiciones de uso.
- Se escribirá documentación de ejemplo mostrando como utilizar cada una de las funciones públicas.
- Se escribirá documentación explicando cómo descargar la biblioteca.
- Se escribirá documentación explicando cómo incluir la biblioteca en un documento HTML para ser ejecutado en un navegador.
- Una vez finalizado el proyecto, se documentarán los requisitos para contribuir al mismo.
- Adicionalmente se escribirá un documento detallando la investigación realizada y las decisiones tomadas, incluyendo:
 - Una introducción al tema, explicando la importancia de la paralelización, los beneficios de los esqueletos algorítmicos entre otras cosas.
 - Análisis de performance de las cada una de las funciones comparadas con implementaciones no paralelas
 - Decisiones y análisis de performance respecto a aspectos como:
 - Serialización de código
 - Partición del trabajo
 - Consolidación de los resultados en orden
 - Devolver los resultados (callbacks o promises)
 - Y en caso de ser implementado:
 - Serialización de Arrays (no Typed Arrays)
 - Serialización de objetos adicionales

Nota: En la sección calendario se presenta el alcance con un mayor nivel de detalle y estimaciones detalladas de cada ítem.

Requerimientos no Funcionales

Hardware

Al momento de escribir este documento los requerimientos de hardware necesarios para instalar el navegador Google Chrome son:

	Windows	Mac	Linux
Procesador	Intel Pentium 4 o	Intel	Intel Pentium 4 o

	superior		superior
Espacio libre en disco	350 MB	350 MB	350 MB
RAM	512 MB	512 MB	512 MB

Fuente: [20]

Adicionalmente, para garantizar el correcto funcionamiento de la biblioteca se necesitara:

- En caso que la biblioteca necesite realizar lecturas/escrituras desde disco, se necesitara un disco de estado sólido (SSD). Al momento de escribir este documento creemos que esto no será necesario, sin embargo durante el transcurso del proyecto podríamos encontrar que esto es un requisito.
- Para poder tomar ventaja de la posibilidad de procesamiento de datos en paralelo se necesitara un procesador con al menos 2 núcleos. Es por eso que como requisito mínimo se deberá contar con un procesador Intel Core i3 o superior.

Software

La biblioteca será desarrollada para su correcto funcionamiento en el navegador Google Chrome, a partir de su versión 38 para sistemas operativos de escritorio (Windows, Unix y OSX).

Esto no quiere decir que la misma no funcione correctamente en otros navegadores, sin embargo esto dependerá del grado de avance de los mismos al momento de completar el trabajo práctico. Por ejemplo, al momento de escribir este documento, solo Google Chrome en su versión 38 (*beta*) permite detectar la cantidad de procesadores disponibles en el dispositivo cliente. Adicionalmente, debido a que gran parte de la naturaleza del trabajo se relaciona con Typed Arrays [12] y Web Workers [9] los cuales dependen de las optimizaciones de los motores de ejecución y considerando que los mismo varían mucho entre cada navegador, no será posible garantizar mejoras de performance del mismo tipo en los diferentes navegadores sin necesidad de un trabajo adicional que excedería el tiempo disponible.

No se probará la biblioteca en dispositivos móviles.

Al momento de escribir este documento, Google Chrome puede ser instalado en los siguiente sistemas operativos:

- Windows XP Service Pack 2+
- Windows Vista
- Windows 7
- Windows 8
- Mac OS X 10.6 o posterior
- Ubuntu 12.04+
- Debian 7+

- OpenSuSE 12.2+
- Fedora Linux 17

Rendimiento

Se seleccionaran uno o dos algoritmos frecuentemente utilizados en aplicaciones que se beneficien de un alto nivel de paralelismo (un ejemplo podría ser la multiplicación de matrices al procesar imágenes). Para cada uno de los mismos se realizaran pruebas de velocidad para mostrar que la utilización de la biblioteca disminuye el tiempo de ejecución de los mismos.

Por otra parte se utilizaran programas de ejemplo los cuales requieran de considerable capacidad de cómputo para verificar el beneficio de utilizar la biblioteca.

El análisis de los resultados será incluido en la documentación del proyecto.

Herramientas de Desarrollo

Hardware

Se utilizarán dos laptops pertenecientes a los alumnos que desarrollan el trabajo práctico.

Laptop 1

Mac Book Pro Retina

Intel Core i7 de 2.8GHz

16 GB memoria RAM DDR3 de 1600 MHz

SSD 250 GB

Laptop 2

Mac Book Air

Intel Core i5 de 1.8GHz

4 GB memoria RAM DDR3 de 1600 MHz

SSD 128 GB

Software

IDE: Sublime Text, Atom.

Lenguaje: EcmaScript versión 6 [15].

Para escribir el documento relacionado con la investigación realizada para crear la librería se utilizará LaTeX [21].

Para escribir la documentación de la librería se utilizará Github Flavored Markdown [22].

Repositorio: Se utilizará un repositorio GIT para control de versiones. El mismo será provisto por GitHub. La URL del proyecto es <https://github.com/pjsteam/pjs>.

Navegador desarrollo/pruebas: Google Chrome (versión > 38). Este navegador (a partir de la mencionada versión) es el único que permite obtener la cantidad de núcleos del dispositivo cliente al momento de escribir este documento [23].

Metodología

La metodología de trabajo será iterativa e incremental, buscando tener productos entregables al final de cada iteración.

Debido a la naturaleza del trabajo (no hay un tiempo fijo dedicado al mismo), se estima que cada alumno trabajará en promedio 12 horas por semana. Consideramos que para proveer entregables de valor, iteraciones de dos semanas permitirán mayores posibilidades de control con nuestro tutor así como también la posibilidad de poder proveer entregables sustanciales.

En la siguiente sección (Cronograma) se indican todos los entregables y la iteración en el que el mismo será presentado.

Cronograma

Aclaraciones:

- Las estimaciones consideran que la documentación será escrita en inglés y no en español.
- Las estimaciones se expresan en horas hombre.
- Las estimaciones usan la distribución beta: $(\text{Min} + 4 \text{ Normal} + \text{Max}) / 6$.
- Se estima aproximadamente un 10% del tiempo para arreglar bugs tanto en el código como documentación.
- Se estima que en promedio cada alumno trabajará 12 horas por semana.

ID	Nombre	Descripción	Mínimo	Normal	Máximo	Estimación
1	Investigación Inicial	Pruebas de concepto para estimaciones				25,0
1.1	Investigación Web Workers	Como funcionan, ejecución dinámica de código	2	3	4	3,0
1.2	Investigación Typed Arrays	Como funcionan, Transferrable Objects	1	2	3	2,0
1.3	Prueba de concepto de map	Utilización de Typed Array y Web Worker con código dinámico para ejecutar map	10	20	30	20,0
2	Infraestructura	Elementos necesarios para				194,7

	biblioteca	la coordinación de la ejecución de las funciones en paralelo				
2.1	Web Workers	Elementos necesarios para manejar a los Web Workers				27,3
2.1.1	Instanciación	Código de la biblioteca para instanciar los Web Workers. Importante teniendo en cuenta el "cold start".	2	4	8	4,3
2.1.2	Implementación	Código del web worker para recibir operaciones a ejecutar	8	16	24	16,0
2.1.3	Destrucción	Código de la biblioteca para destruir los Web Workers.	2	4	6	4,0
2.1.4	Documentación de las operaciones	Documentar en la documentación de la biblioteca	2	3	4	3,0
2.2	Pruebas de concepto serialización de código	Verificar la forma más rápida para pasar funciones entre hilo principal y Web Worker				48,0
2.2.1	Investigación	Realizar una búsqueda para determinar las alternativas posibles	16	24	32	24,0
2.2.2	Pruebas	Realizar pruebas de performance analizar cada alternativa	12	16	32	18,0
2.2.3	Documentación de resultados	Incluir el analisis en el documento de investigación	4	6	8	6,0
2.3	Tranferencia de código y typed array	Encargado de enviar código a Web Workers y datos para su procesamiento.				27,0
2.3.1	Investigar alternativas partición del Typed Array	Determinar la mejor manera (en cuanto a performance) para particionar el Typed Array a ser distribuido entre diferentes Web Workers	12	16	24	16,7
2.3.2	Implementación	Código para particionar el Typed Array	2	4	8	4,3
2.3.3	Documentación de resultados de la investigación	Documentar en el documento de investigación los resultados	4	6	8	6,0
2.4	Consolidación de resultados	Asegurar que el typed array a devolver este en el mismo orden que el original				22,3
2.4.1	Investigar alternativas para consolidar los resultados	Determinar la forma más rápida de guardar información de cada parte y volver a unir las	8	12	16	12,0
2.4.2	Implementación	Código para persistir el orden de las partes y volver a	2	4	8	4,3

		unirlas				
2.4.3	Documentación de resultados de la investigación	Documentar en el documento de investigación los resultados	4	6	8	6,0
2.5	Empaquetamiento	Aspectos relacionados con la modularización de la biblioteca				24,3
2.5.1	Selección sistema de módulos	Determinar el tipo de sistema de módulos a utilizar (AMD, CommonJS, EcmaScript 6)	8	16	24	16,0
2.5.2	Implementación	Actualizar el código de la librería para implementar el sistema de módulos seleccionados	2	8	16	8,3
2.6	Soporte para Generics	Soportar el procesamiento de cualquier tipo de Typed Array	2	4	8	4,3
2.7	Retorno de resultados	Relacionado con la forma de devolver el resultado asincrónicamente				25,3
2.7.1	Investigar alternativas	Promises o callbacks. Utilizar una biblioteca de Promises o las nativas.	4	16	24	15,3
2.7.2	Implementación	Implementar el código para devolver los resultados	2	4	6	4,0
2.7.3	Documentación de resultados de la investigación	Documentar en el documento de investigación los resultados	4	6	8	6,0
2.8	Encolar operaciones pendientes	Funcionalidad para encolar nuevas operaciones si hay pendientes, relacionado con la naturaleza asincrónica de la operación.	8	16	24	16,0
3	Funciones de Orden Superior	Tareas relacionadas con cada una de las funciones				207,3
3.1	Map	Tareas relacionadas con la función Map				73,3
3.1.1	Pruebas unitarias	Código para verificar el funcionamiento de forma automática	8	16	20	15,3
3.1.2	Implementación	Implementación de la función	12	16	20	16,0
3.1.3	Benchmarks	Pruebas de performance comparada con la implementación secuencial	8	16	32	17,3
3.1.4	Encadenamiento de funciones	Soportar llamadas consecutivas a esta u otras funciones sin la necesidad de materializar los resultados	4	12	24	12,7
3.1.5	Documentación de la función	Documentar la función en la documentación de la	2	4	6	4,0

		biblioteca				
3.1.6	Documentación de los resultados	Documentar los resultados en el documento de investigación	4	8	12	8,0
3.2	Filter	Tareas relacionadas con la función Filter				73,3
3.2.1	Pruebas unitarias	Código para verificar el funcionamiento de forma automática	8	16	20	15,3
3.2.2	Implementación	Implementación de la función	12	16	20	16,0
3.2.3	Benchmarks	Pruebas de performance comparada con la implementación secuencial	8	16	32	17,3
3.2.4	Encadenamiento de funciones	Soportar llamadas consecutivas a esta u otras funciones sin la necesidad de materializar los resultados	4	12	24	12,7
3.2.5	Documentación de la función	Documentar la función en la documentación de la biblioteca	2	4	6	4,0
3.2.6	Documentación de los resultados	Documentar los resultados en el documento de investigación	4	8	12	8,0
3.3	Reduce	Tareas relacionadas con la función Reduce				60,7
3.3.1	Pruebas unitarias	Código para verificar el funcionamiento de forma automática	8	16	20	15,3
3.3.2	Implementación	Implementación de la función	12	16	20	16,0
3.3.3	Benchmarks	Pruebas de performance comparada con la implementación secuencial	8	16	32	17,3
3.3.4	Documentación de la función	Documentar la función en la documentación de la biblioteca	2	4	6	4,0
3.3.5	Documentación de los resultados	Documentar los resultados en el documento de investigación	4	8	12	8,0
4	Benchmarks generales	Pruebas de performance aplicadas a algoritmos que requieren un alto nivel de procesamiento				52,0
4.1	Selección algoritmos para comparación	Selección de dichos algoritmos (por ejemplo: procesamiento de imágenes)	8	16	24	16,0
4.2	Ejecución de las pruebas	Implementación y ejecución del benchmark	8	24	40	24,0
4.3	Documentación	Documentar los resultados	8	12	16	12,0

	resultados	en el documento de investigación				
5	Utilización de Arrays no Typed Arrays	Extensión de la biblioteca para que soporte el uso de Arrays de javascript además de los Typed Arrays				94,7
5.1	Investigar alternativas para la implementación	Determinar la mejor manera (en cuanto a performance) para el soporte de los Typed Arrays (transferencia hacia los Web Workers, Consolidación)	16	24	40	25,3
5.2	Realizar pruebas de performance	Código de las pruebas de performance en comparación con las versiones en serie de las funciones.	8	16	24	16,0
5.3	Documentación resultados	Documentar los resultados en el documento de investigación.	4	8	16	8,7
5.4	Implementación (opcional)	En caso de que sea factible la extensión de la biblioteca se procederá a implementar la misma.				44,7
5.4.1	Código	Código para soporte de Arrays.	8	16	32	17,3
5.4.2	Pruebas	Código que verifica el correcto funcionamiento del uso de Arrays.	8	16	24	16,0
5.4.3	Documentación de uso	Extender la documentación de la biblioteca en relación al impacto del soporte de Arrays en cada función.	6	12	14	11,3
6	Pasaje de datos adicionales	Extensión de la biblioteca para que soporte el pasaje de datos adicionales para solventar la pérdida de closures.				124,0
6.1	Investigar alternativas para la implementación	Determinar la mejor manera (en cuanto a performance) para el soporte de pasaje de parámetros adicionales.	16	24	40	25,3
6.2	Realizar pruebas de performance	Código de las pruebas de performance en comparación con la versión que no envía estos nuevos parámetros.	8	16	24	16,0
6.3	Documentación resultados	Documentar los resultados en el documento de investigación.	4	8	16	8,7
6.4	Implementación	Implementación de las adiciones a la biblioteca				74,0
6.4.1	Código	Código para soporte de	24	40	48	38,7

		pasaje de parámetros adicionales.				
6.4.2	Pruebas	Código que verifica el correcto funcionamiento del pasaje de parámetros adicionales.	12	24	32	23,3
6.4.3	Documentación de uso	Extender la documentación de la biblioteca en relación al impacto del soporte de parámetros adicionales en cada función.	8	12	16	12,0
7	Documentación adicional	Documentación adicional a crear no cubierta en los ítems previos				25,8
7.1	Introducción al documento de investigación	El documento de investigación tendrá una introducción explicando las razones por las que el paralelismo es beneficioso, el porqué de la utilización de esqueletos algorítmicos y funciones de alto orden.	4	8	10	7,7
7.2	Guías para contribuir al proyecto una vez finalizado	Documento público que explicará los pasos a seguir para contribuir al proyecto open source una vez finalizado el trabajo profesional.	2	4	8	4,3
7.3	Presentación final del trabajo	Presentación utilizada para mostrar el trabajo realizado cuando el mismo esté listo para la entrega.				11,7
7.3.1	Creación diapositivas		4	8	10	7,7
7.3.2	Ensayo		2	4	6	4,0
7.4	Descargar y utilización de la biblioteca	Documentación pública explicando cómo descargar la biblioteca y utilizarla en un sitio web.	1	2	4	2,2
8	Bug fixing y otras modificaciones					80,0
	Total					803,5

A continuación se detallan los entregables de cada iteración, haciendo referencia a los ítems de la tabla anterior.

1	2
1. Investigación Inicial	2.2.2 Pruebas serialización Código

2.2.1 Investigación - Serialización Código	2.1.1 Instanciación WWW
	2.1.3 Destrucción WW
	2.1.4 Documentación funciones WW
	2.3.1 Investigar alternativas partición del Typed Array

3	4
2.1.2 Implementación WW	2.4.3 Documentación de la investigación - consolidación resultados
2.2.3. Documentación de resultados serialización código	3.1.2 Implementación - Map
2.3.2 Implementación partición typed array	3.1.1 Pruebas unitarias - Map
2.3.3 Documentación resultados partición Typed Array	3.1.5 Documentación función - Map
2.4.1 Investigación alternativas consolidación resultados	2.6 Soporte para generics
2.4.2 Implementación alternativas consolidación resultados	

5	6
2.5 Empaquetamiento	3.2.1 Pruebas Unitarias - Filter
3.1.3 Benchmarks - Map	3.2.2 Implementación - Filter
3.1.6 Documentación Resultados - Map	3.2.3 Benchmarks - Filter

7	8
3.2.5 Documentación función - Filter	3.1.4 Encadenamiento funciones - Map
3.2.6 Documentación resultados - Filter	3.2.4 Encadenamiento funciones - Filter
3.3.1 Pruebas unitarias - Reduce	3.3.3 Benchmarks - Reduce
3.3.2 Implementación - Reduce	3.3.5 Documentación resultados - Reduce
3.3.5 Documentación función Reduce	

9	10
6.1. Investigar alternativas para la implementación - Datos Adicionales	7.1 Introducción Documento Investigación
6.2. Pruebas Performance - Datos Adicionales	7.2 Guías contribución proyecto
6.3. Documentación Resultados - Datos Adicionales	2.8 Encolar operaciones pendientes

11	12
6.4 Implementación - Datos Adicionales	4 - Benchmarks generales

13	14
6.1. Investigar alternativas para la implementación - Not Typed Arrays	5.4 Implementación - Not Typed Arrays
6.2. Pruebas Performance - Not Typed Arrays	
6.3. Documentación Resultados - Not Typed Arrays	

15	16	17
2.7 Retorno resultados	Bug fixing	Bug fixing
7.4 Documentación descarga y utilización biblioteca		7.3 Presentación final
Bug fixing		

Bibliografía

[1] G. E. Moore, "Cramming More Components onto Integrated Circuits".

[2] ITRS. [Online]. Available:

http://www.itrs.net/Links/2010ITRS/2010Update/ToPost/2010Tables_ORTC_ITRS.xls.

- [3] Microsoft Corp, "Parallel Programming in the .NET Framework," [Online]. Available: [http://msdn.microsoft.com/en-us/library/dd460693\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/dd460693(v=vs.110).aspx).
- [4] ECMA, "EcmaScript," [Online]. Available: <http://www.ecmascript.org/>.
- [5] Unknown, "asm.jq," [Online]. Available: <http://asmjs.org/>.
- [6] Khronos Group, [Online]. Available: <http://www.khronos.org/webgl/>.
- [7] J. Wang, N. Rubin and S. Yalamanchili, "ParallelJS: An Execution Framework for JavaScript on Heterogeneous Systems".
- [8] A. Rauschmayer, "JavaScript gains support for SIMD," [Online]. Available: <http://www.2ality.com/2013/12/simd-js.html>.
- [9] Refsnes Data, "HTML5 Web Workers," [Online]. Available: http://www.w3schools.com/html/html5_webworkers.asp.
- [10] W3C, «Web IDL,» [En línea]. Available: <http://www.w3.org/TR/WebIDL/#Clamp>.
- [11] IEEE, «754-2008 - IEEE Standard for Floating-Point Arithmetic,» 2008.
- [12] Mozilla Developer Network and individual contributors, «JavaScript typed arrays,» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays.
- [13] Mozilla Developer Network and individual contributors, «Array,» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array.
- [14] E. Bidelman, «Transferable Objects: Lightning Fast!,» [En línea]. Available: <http://updates.html5rocks.com/2011/12/Transferable-Objects-Lightning-Fast>.
- [15] ECMA, «ECMA-262 6th Edition Draft,» 24 Agosto 2014. [En línea]. Available: <https://people.mozilla.org/~jorendorff/es6-draft.html#sec-operations-on-promise-objects>.
- [16] Mozilla Developer Network and individual contributors, "Object.isFrozen()," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/isFrozen.
- [17] amdjs organization, "amdjs API," [Online]. Available: <https://github.com/amdjs/amdjs-api/>.
- [18] CommonJS Organization, "CommonJS," [Online]. Available: <http://wiki.commonjs.org/wiki/CommonJS>.

- [19] Ecma, "Harmony Specification Drafts," [Online]. Available: http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts.
- [20] Google, "Chrome system requirements," [Online]. Available: <https://support.google.com/chrome/answer/95411?hl=en>.
- [21] LaTeX Organization, "LaTeX – A document preparation system," [Online]. Available: <http://www.latex-project.org>.
- [22] GitHub, "Github Flavored Markdown," [Online]. Available: <https://help.github.com/articles/github-flavored-markdown>.
- [23] WHATWG Wiki, "Navigator HW Concurrency," [Online]. Available: https://wiki.whatwg.org/wiki/Navigator_HW_Concurrency.
- [24] 14, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays.
- [25] 15, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays.
- [26] 16, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Guide/Performance/Using_web_workers.
- [27] 17, [Online]. Available: http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts.

Curricululo/s Autor/es

Alumno: Matías Servetto

Padrón: 91363

Plan: 1986

Carrera: 10 – Ingeniería en Informática

Promedio (con/sin aplazos): 7.50

Materia	Créditos	Fecha	Nota	Acta o Resolución
(6103) ANALISIS MATEMATICO II A	8	28/12/2009	8	61-154-139
(6201) FISICA I A	8	11/02/2010	5	62-108-24

(7540) ALGORITMOS Y PROGRAMACION I	6	09/02/2010	7	95-102-240
(6108) ALGEBRA II A	8	23/02/2011	5	61-158-38
(6203) FISICA II A	8	15/07/2010	6	62-108-116
(6301) QUIMICA	6	16/07/2010	7	63-75-70
(7541) ALGORITMOS Y PROGRAMACION II	6	20/07/2010	8	95-104-104
(6215) FISICA III D	4	20/07/2011	8	62-109-82
(6602) LABORATORIO	6	04/03/2011	7	86-140-115
(6670) ESTRUCTURA DEL COMPUTADOR	6	29/06/2011	5	86-140-134
(7507) ALGORITMOS Y PROGRAMACION III	6	14/12/2010	9	95-105-110
(7512) ANALISIS NUMERICO I	6	27/06/2011	8	95-106-137
(6109) PROBABILIDAD Y ESTADISTICA B	6	14/12/2011	8	61-159-106
(6110) ANÁLISIS MATEMÁTICO III A	6	16/02/2012	7	61-157-112
(6620) ORGANIZACION DE COMPUTADORAS	6	23/02/2012	6	86-141-197
(7506) ORGANIZACION DE DATOS	6	05/12/2011	8	95-107-209
(7542) TALLER DE PROGRAMACION I	4	21/12/2011	9	95-108-89
(7112) ESTRUCTURA DE LAS ORGANIZACIONES	6	04/07/2012	6	71-154-49
(7114) MODELOS Y OPTIMIZACION I	6	18/07/2012	10	71-154-112
(7508) SISTEMAS OPERATIVOS	6	20/07/2012	8	95-109-155
(7509) ANALISIS DE LA INFORMACION	6	02/07/2012	7	95-109-46
(7510) TECNICAS DE DISEÑO	6	10/12/2012	8	95-110-128
(7515) BASE DE DATOS	6	19/12/2012	7	95-110-227
(7543) INTRODUCCION A LOS SISTEMAS DISTRIBUIDOS	6	14/12/2012	9	95-110-176
(7552) TALLER DE PROGRAMACION II	4	15/02/2013	10	95-111-93
(7140) LEG. Y EJ. PROF. DE LA ING. EN INFORMAT.	4	13/12/2013	8	71-0001461
(7113) INFORMACION EN LAS ORGANIZACIONES	6	11/02/2014	6	71-0001256
(7544) ADM. Y CONTROL DE PROY. INFORMATICOS I	6	23/07/2013	4	95-0001248

(7545) TALLER DE DESARROLLO DE PROYECTOS I	6	08/08/2013	9	95-0001357
(7546) ADM. Y CONTROL DE PROY. INFORMATICOS II	6	04/02/2014	8	95-0001434
(7547) TALLER DE DESARROLLO DE PROYECTOS II	6	13/12/2013	8	95-0001659
(7548) CALIDAD EN DESARROLLO DE SISTEMAS	4	07/07/2014	6	95-0002029
(6107) MATEMATICA DISCRETA	6	21/12/2010	7	61-156-67
(7559) TECNICAS DE PROGRAMACION CONCURRENTE I	6	19/07/2013	9	95-0001151
(7567) SIST.AUTOM.DE DIAG.Y DETEC.DE FALLAS I	6	29/07/2013	8	95-0001287
(7569) SIST.AUTOM.DE DIAG.Y DETEC.DE FALLAS II	6	17/02/2014	8	95-0001749
(7526) SIMULACION	6	26/02/2014	7	95-0001867
(7565) MANUFACTURA INTEGRADA POR COMP.(CIM) I	6	30/06/2014	6	95-0001993
(6669) CRIPTOGRAFIA Y SEGURIDAD INFORMATICA	6	21/07/2014	10	86-0001781
(7550) INTROD. A LOS SISTEMAS INTELIGENTES	6		10	

Alumno: Damián Schenkelman

Padrón: 90728

Plan: 1986

Carrera: 10 – Ingeniería en Informática

Promedio (con/sin aplazos): 8.00

Materia	Fecha	Nota	Acta o Resolución
(6103) ANALISIS MATEMATICO II A	6/8/09	6	61-154-71
(6201) FISICA I A	25/08/2009	6	62-107-201
(7540) ALGORITMOS Y PROGRAMACION I	9/10/09	10	95-102-93

(7541) ALGORITMOS Y PROGRAMACION II	28/12/2009	9	95-102-185
(6203) FISICA II A	25/02/2010	7	62-108-62
(6108) ALGEBRA II A	3/3/10	5	61-153-177
(6301) QUIMICA	5/7/10	8	63-75-69
(7507) ALGORITMOS Y PROGRAMACION III	13/07/2010	10	95-104-47
(6110) ANÁLISIS MATEMÁTICO III A	4/8/10	4	61-157-18
(7512) ANALISIS NUMERICO I	15/12/2010	8	95-105-131
(6670) ESTRUCTURA DEL COMPUTADOR	22/12/2010	10	86-140-10
(6109) PROBABILIDAD Y ESTADISTICA B	10/2/11	4	61-159-1
(7801) IDIOMA INGLES	17/02/2011	9	78-23-75
(7542) TALLER DE PROGRAMACION I	29/06/2011	9	95-106-157
(6215) FISICA III D	20/07/2011	9	62-109-82
(6602) LABORATORIO	28/07/2011	9	86-140-224
(7506) ORGANIZACION DE DATOS	21/12/2011	9	95-108-94
(6620) ORGANIZACION DE COMPUTADORAS	6/2/12	8	86-141-149
(7114) MODELOS Y OPTIMIZACION I	22/02/2012	7	71-153-193
(7509) ANALISIS DE LA INFORMACION	2/7/12	6	95-109-46
(7508) SISTEMAS OPERATIVOS	12/7/12	10	95-109-103
(7112) ESTRUCTURA DE LAS ORGANIZACIONES	18/07/2012	7	71-154-113
(7552) TALLER DE PROGRAMACION II	17/08/2012	10	95-110-87
(7510) TECNICAS DE DISEÑO	10/12/12	9	95-110-128
(7543) INTRODUCCION A LOS SISTEMAS DISTRIBUIDOS	14/12/2012	9	95-110-176
(7515) BASE DE DATOS	19/12/2012	10	95-110-227
(7544) ADM. Y CONTROL DE PROY. INFORMATICOS I	23/07/2013	5	95-0001248
(7567) SIST.AUTOM.DE DIAG.Y DETEC.DE FALLAS I	29/07/2013	8	95-0001287

(7559) TECNICAS DE PROGRAMACION CONCURRENT E I	30/07/2013	10	95-0001286
(7545) TALLER DE DESARROLLO DE PROYECTOS I	8/8/13	9	95-0001357
(7546) ADM. Y CONTROL DE PROY. INFORMATICOS II	10/12/13	7	95-0001489
(7140) LEG. Y EJ. PROF. DE LA ING. EN INFORMAT.	13/12/2013	8	71-0001461
(7547) TALLER DE DESARROLLO DE PROYECTOS II	13/12/2013	8	95-0001659
(7113) INFORMACION EN LAS ORGANIZACIONES	17/12/2013	7	71-0001375
(7569) SIST.AUTOM.DE DIAG.Y DETEC.DE FALLAS II	17/02/2014	8	95-0001749
(7565) MANUFACTURA INTEGRADA POR COMP.(CIM) I	30/06/2014	7	95-0001993
(6669) CRIPTOGRAFIA Y SEGURIDAD INFORMATICA	21/07/2014	10	86-0001781
(7570) SIST.DE PROG. NO CONVENCIONAL DE ROBOTS	4/8/14	9	95-0002264

Fuente: Sistema GUARANI.

Nota: Al día 4 de Octubre del 2014, la materia 75.50 no se encuentra en el sistema GUARANI

Información Personal

Matias Eduardo Servetto

1989-07-04

Ecuador 871, El Talar, Tigre, Buenos Aires, Argentina

Celular: (0054) 011-3-757-4076

E-mail: servetto.matias@gmail.com

Experiencia Laboral

Desarrollador en Creative Coefficient S.A. (2013 – 2014)

Desarrollador de aplicaciones para dispositivos móviles.

Analista/Desarrollador en Despegar.com, Inc (2014-Presente)

Miembro del equipo iOS en el equipo Mobile

Información Personal

Damian Ezequiel Schenkelman

1990-04-09

Formosa 252 1A, Capital Federal, Argentina

Casa: (0054) 011-4-903-9499 / Celular: (0054) 011-6-288-8313

E-mail: damian.schenkelman@gmail.com

Experiencia Laboral

Desarrollador en Auth0 (Mayo 2014 – Presente)

Me desempeño primordialmente desarrollando aplicaciones en el *backend* del producto.

Desarrollador en Microsoft (pasantía) – Redmond WA (Jan 2014 – Mar 2014)

Parte del equipo de "deployment automation" dentro de Azure SQL Database.

Desarrollador en Southworks (Marzo 2008 – Mayo 2014)

Desarrollador y líder de proyecto en varios proyectos con Microsoft como cliente.

Ayudante ad-honorem en FIUBA (Enero 2010 – Febrero 2011)

Materias Algoritmos I y Algoritmos II, cátedra de Rosa Wachenhauzer.

Plan de Cursado

El alumno Schenkelman, Damián tiene 232 créditos y se encuentra actualmente (2do Cuatrimestre 2014) cursando la materia 75.48 "Calidad en el Desarrollo de Sistemas" la cuál es la última materia pendiente por aprobar (sin considerar el presente trabajo).

El alumno Servetto, Matías tiene 238 créditos. No debe cursar ninguna materia.