DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 3

Algoritmos en sistemas distribuidos

13 de noviembre de 2014

Sistemas Operativos

Integrante	LU	Correo electrónico
Rodriguez, Pedro	197/12	pedro3110.jim@gmail.com
Benegas, Gonzalo Segundo	958/12	gsbenegas@gmail.com
Barrios, Leandro Ezequiel	404/11	ezequiel.barrios@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Introducción

Hoy en día, la globalización y creciente utilización mundial masiva de medios de información como la internet, ha impulsado la existencia de gigantes de la información. Ejemplos de estos son empresas como Facebook, Yahoo!, Google, Twitter y otros. La mayor parte del servicio provisto por estas empresas a sus usuarios, consiste y necesita de la utilización, procesamiento y análisis de grandes bases de datos. Por ello, una de las más frecuentes acciones que deben ser efectuadas es el filtrado de datos y su posterior procesamiento.

Un método comúnmente utilizado dentro del entorno de procesamiento de datos a través de clusters es el **MapReduce**, creado por la empresa Google. Este permite analizar un gran conjunto de datos, relacionándolos a través de un índice (*key*) a través de una operación `map`, para luego operar sobre los conjuntos de valores asociados a cada *key* a través de una operación `reduce`. Opcionalmente, es posible aplicar al conjunto de datos producido por `reduce`, una última operación `finalize`.

Considerando la inmensidad del volumen de información con el que se trabaja, y el gran costo económico que implican la fabricación y el funcionamiento de los servidores adecuados para manipularla, es importante lograr el máximo aprovechamiento de los recursos computacionales de los que estas empresas disponen. Por ello, se intenta trabajar con la máxima eficiencia algorítmica, temporal y de espacio posibles.

Contents

Parte I Implementación de Map-Reduce

1.1 Encontrar el subreddit con mayor score promedio

Figure 1.1.1: La función `map`

```
function () {
    emit(this.subreddit, { count: 1, sum: this.score} );
}
```

La función *map* se aplica a cada post y emite, con el *subreddit* como key, una tupla con la cantidad de posts - inicializada en 1 - y la suma de los scores – inicializada con el score del post.

Figure 1.1.2: La función `reduce`

```
function (key, values) {
    reducedVal = {count: 0, sum: 0};
    for (var i = 0; i < values.length; i++) {
        reducedVal.count += values[i].count;
        reducedVal.sum += values[i].sum;
    };
    return reducedVal;
}
```

Tras aplicar la función *map* quedan asociados cada subreddit con una lista de las tuplas asociados a los posts del subreddit. La función *reduce* toma un fragmento de esta lista y la reduce en una sola tupla que acumula la cantidad de posts y la suma de los scores.

Figure 1.1.3: La función `finalize`

```
function finalize (key, reducedVal) {
    return reducedVal.sum / reducedVal.count;
}
```

Luego de la etapa de *reduce*, quedan asociados cada subreddit con su cantidad de posts y su suma de scores. La función $finalize$ calcula el promedio final realizando una división.

Ejecutamos el query y utilizamos la función `sort` en la consola de Mongo. El subreddit *GirlGamers* tiene el mayor score promedio: 2483.

1.2 Encontrar los doce títulos con mayor score de la colección de posts con al menos 2000 votos.

Figure 1.2.4: La función `map`

```
function () {
    if (this.total_votes >= 2000)
        emit(this.title, this.total_votes);
}
```

Se emiten, para todos los posts con al menos 2000 votos, su título y su cantidad de votos.

Figure 1.2.5: La función `reduce`

```
function (key, values) {
    return Array.sum(values);
}
```

Para cada título, se suman los votos de sus posts.

Tras ejecutar el query, se ordenaron los resultados. Los doce títulos con mayor score de la colección de posts con al menos 2000 votos son:

Título	**Score**
"Airline screwed up, a friend just posted this on Facebook."	177103
"Following the Obama AMA"	144145
"Nailed it."	129307
"Help a brother out!"	129183
"Seen in NJ, what a friendly neighbor"	126222
"McKayla Maroney visits the White House... her picture with the President"	120929
"The Bus Knight"	119774
"Brilliant and thoughtful parents handed these out to everyone on my flight."	119232
"Screenshot of reddit from the year 3012"	118866
"Standing guard, hurricane or otherwise"	117272
"My neighbors are taking this especially hard."	117101
"When I found out I could upvote by pressing 'A"'	115717

1.3 Para los diez mejores scores, calcular la cantidad de comentarios en promedio por sumisión.

Este ejercicio se resuelve de manera análoga al ejercicio 1.

Figure 1.3.6: La función `map`

```
function () {
    emit(this.score, {count: 1, sum: this.number_of_comments});
}
```

Figure 1.3.7: La función `reduce`

```
function (key, values) {
    reducedVal = {count: 0, sum: 0};
    for (var i = 0; i < values.length; i++) {
        reducedVal.count += values[i].count;
        reducedVal.sum += values[i].sum;
    };
    return reducedVal;
}
```

Figure 1.3.8: La función `finalize`

```
function finalize (key, reducedVal) {
    return reducedVal.sum / reducedVal.count;
}
```

Los resultados se ordenaron. Los diez mejores scores y la cantidad de comentarios en promedio por sumisión son:

Score	**Comentarios en promedio**
20570	1463
12333	1612
11908	2681
10262	1514
8935	480
8835	1716
8699	934
8241	571
7297	1110
6741	2204

1.4 Entre los usuarios con a la sumo 5 sumisiones, encontrar el que posea mayor cantidad de upvotes.

Este ejercicio se resuelve de forma similar al ejercicio 1, salvo que en la función *finalize* se filtran los usuarios con más de 5 sumisiones.

Figure 1.4.9: La función `map`

```
function () {
    emit(this.username, {sumisiones: 1, number_of_upvotes: this.number_of_upvotes});
}
```

Figure 1.4.10: La función `reduce`

```
function (key, values) {
    var res = {sumisiones: 0, number_of_upvotes: 0};
    for (var i = 0; i < values.length; i++) {
        var val = values[i];
        res.sumisiones += val.sumisiones;
        res.number_of_upvotes += val.number_of_upvotes;
    }
    return res;
}
```

Figure 1.4.11: La función `finalize`

```
function finalize (key, reducedVal) {
    if (reducedVal.sumisiones <= 5)
        return reducedVal.number_of_upvotes;
}
```

Los resultados se ordenaron. El usuario con a lo sumo cinco sumisiones ya la mayor cantidad de upvotes es “lepry”, con 90396 upvotes.

1.5 Para todos los subrredit que poseen un score entre 280 y 300, indicar la cantidad palabras presentes nnen sus títulos

Este ejercicio se resuelve de forma similar al ejercicio 1, salvo que en la función *finalize* se filtran los subreddit cuyo score no se encuentra en el intervalo [280,300].

Figure 1.5.12: La función `map`

```
function () {
    var cantPalabras = this.title.split(' ').length;
    emit(this.subreddit, {score: this.score, cantPalabras: cantPalabras});
}
```

Figure 1.5.13: La función `reduce`

```
function (key, values) {
    var res = {score: 0, cantPalabras: 0};
    for (var i = 0; i < values.length; i++) {
        var val = values[i];
        res.score += val.score;
        res.cantPalabras += val.cantPalabras;
    }
    return res;
}
```

Figure 1.5.14: La función `finalize`

```
function finalize (key, reducedVal) {
    if (280 <= reducedVal.score && reducedVal.score <= 300)
        return reducedVal.cantPalabras;
}
```

Los únicos resultados arrojados fueron:

Subreddit	**Palabras en títulos**
"Firearms"	24
"anime"	18
"Sexy"	13
"Feminism"	6
"HeroesofNewerth"	6
"TheRealZachAnner"	4
"ragecomics"	4
"xkcd"	4

Parte II Investigación

Para esta parte del TP se pide leer e interpretar el paper «**Job Scheduling for Multi- User MapReduce Clusters**»[1]. Se sugiere además una serie de puntos para analizar el contenido del paper. Estos son los siguientes:

2.1 Explique cuál es el entorno y la situación donde se plantea el problema. (Motivación)

2.2 Identifique de problema

2.3 Identifique situaciones donde el problema se dispara y la consecuencias que esto genera.

2.4 Comente el background histórico del problema

2.5 Explique otros problemas asociados a la búsqueda de una mejor solución

2.6 Explique las soluciones propuestas

2.7 Evalúe soluciones propuestas

2.8 Discusión

2.9 Conclusiones

[1] http://www.icsi.berkeley.edu/pubs/techreport s/ICSI_jobschedulingfor09.pdf

2.1 Explique cuál es el entorno y la situación donde se plantea el problema. (Motivación)

La idea tras este paper surgió luego de que los autores se vieran frente a la tarea de diseñar un scheduler para MapReduce, para la empresa Facebook. Este sería el encargado de repartir el poder de cómputo en un *warehouse Hadoop*[2] de 600 nodos, en un entorno multiusuario, en donde se combinarían tareas de producción con tareas experimentales.

Este warehouse era utilizado para aproximadamente 3200 operaciones de MapReduce diarias, siendo que algunas de ellas eran tareas frecuentes de mantenimiento, análisis de datos, antispam y optimización, cuya ejecución se prolongaba a lo largo de horas, mientras que otras eran queries *AD-HOC* cuya ejecución demoraba unos pocos minutos. Por tal motivo, era de vital importancia poder lograr un mecanismo de **scheduling justo**.

[2]Hadoop es la versión *open-source* de MapReduce.

2.2 Identifique el problema

Se describen principalmente dos aspectos en donde el scheduling de un cluster MapReduce se diferencia de un mecanismo genérico de scheduling, y por los cuales el mecanismo de scheduling utilizado previamente presentaba una pérdida de rendimiento de entre 2 y 10 veces en comparación con el mecanismo presentado en el paper. Los dos principales aspectos problemáticos fueron la *localidad de datos*, y la *interdependencia entre las operaciones Map y Reduce*.

Localidad de Datos

Este problema comprende la necesidad inherente de la técnica MapReduce de tener acceso local a los datos con los cuales se están trabajando. Esta necesidad se debe principalmente a que a diferencia de los algoritmos "`CPU- INTENSIVE`"[3], el MapReduce entra en la categoría de los denominados "`DATA-INTENSIVE`", es decir, aquellos en donde el poder de cómputo necesario es despreciable frente a la gran cantidad de datos con los que se trabaja.

Desde esta perspectiva, contar con un scheduler eficiente en el acceso y la utilización de los datos, es mucho más importante que en otro tipo de situaciones. Todo esto está agravado por el costo que supone el almacenamiento, y la transmisión de datos entre distintos servidores. Este defecto se puede observar principalmente en **trabajos pequeños y/o concurrentes**.

Interdependencia

Se entiende por interdependencia a la necesidad de que todas las tareas de `map` se encuentren finalizadas al momento de iniciar la ejecución de `reduce`. Es fácil percibir la gran cantidad de inconvenientes que esto genera. Se mencionan particularmente los casos de **infrautilización de los recursos y `starvation`**, en donde se da que una única tarea adquiere cierta cantidad de slots, destinados a realizar la operación `reduce`, pero que se ven bloqueados/inutilizados hasta el momento en que esta tarea termine de realizar su operación de `map`, evitando de esta forma que otras tareas se ejecuten, y de **consumo excesivo de espacio de disco** destinado a los datos intermedios producidos por `map` (datos que no pueden ser liberados hasta que el trabajo termine).

[3] Los que necesitan un gran poder de cómputo.

2.3 Identifique situaciones donde el problema se dispara y la consecuencias que esto genera.

Los problemas descriptos anteriormente se ponían en evidencia al momento de querer realizar queries `ad-hoc` de muy corta duración frente a los trabajos de producción que el warehouse debe correr periódicamente. Bajo esta perspectiva, era deseable que los trabajos experimentales puedan ser lanzados en cualquier momento, y tener un tiempo de respuesta **aceptable**. Con la implementación del Scheduler FIFO de Hadoop, esto se volvía imposible, reduciendo significativamente la utilidad del sistema.

FIX-ME

2.4 Comente el background histórico del problema

Hadoop está inspirado en el MapReduce de Google, por lo que gran parte de su implementación básica está fuertemente ligada a este proyecto. La primer solución al conflicto del scheduling, se encuentra provista por defecto dentro la propia implementación de Hadoop, y consta de una cola `FIFO` de 5 niveles de prioridad, de forma tal que cada vez que un slot se libera, el scheduler lo asigna a la más prioritaria de entre las tareas pendientes.

Sobre este enfoque, Hadoop aplica una `optimización de localidad`, al igual que lo hace MapReduce de Google: dado que una tarea generalmente consta de múltiples operaciones de `map`, luego de elegir una tarea el scheduler selecciona las operaciones map más adecuadas según un **criterio de localidad**. Serán elegidos entonces los maps que utilicen datos que se encuentren más cerca físicamente al worker que está corriendo la tarea. Es decir, serán elegidas primero las que se encuentren en ese mismo worker, luego las que se encuentren en el mismo rack, y finalmente las que se encuentren en racks remotos.

La gran desventaja del scheduler `FIFO` es el pésimo tiempo de respuesta para trabajos pequeños cuando se encuentran intercalados con trabajos grandes. Por ello, la primer solución que se implementó, es el mecanismo denominado **Hadoop On Demand (HOP)**.

FIX-ME

2.5 Explique otros problemas asociados a la búsqueda de una mejor solución

Para explicar el problema, es necesario ahondar un poco en los conceptos utilizados en el paper. Se describe un `job` como un conjunto de `tasks`, siendo un task una operación `map-reduce`. Cada nodo (también llamados `slaves` o `workers`) tiene una determinada cantidad de `slots de ejecución`, y conforme estos se van liberando el Scheduler le va asignando `tasks`, siendo que cada uno de estos ocupa un `slot`.

Problemas asociados a la Localidad de Datos: **Head-of-line scheduling**: Para explicar este concepto, el paper propone una métrica, **el porcentaje de localidad de un job en relación a su tamaño**. Utilizando esta métrica, dado un `job`, su porcentaje de localidad se ve representado por la cantidad de `nodos` en la que se encuentran distribuídos los datos de sus `tasks`, en relación al total.

Sticky Slots:

Problemas asociados a la Interdependencia de Map-Reduce:

2.6 Explique las soluciones propuestas

2.7 Evalúe soluciones propuestas

2.8 Discusión

2.9 Conclusiones