BIG DATA FRAMEWORK

Google Cluster Data 2011 - 2

Élèves :
Sordel Gabin
Manach Titouan
Bellegarde Clement
Rhihil Aymen
Mazeau Leo

Professeur :
Del Castillo
Juan Angel Lorenzo

11 février 2024

Table des matières

1 Introduction

La trace d'utilisation d'un cluster de Google représente un ensemble de données de grande valeur, collectées par Google sur l'un de leurs clusters pendant le mois de mai 2011. Avec une taille considérable de 41 GBytes, ces données ont été mises à disposition de la communauté scientifique dans le but d'encourager la recherche sur l'analyse des données des grands centres de calcul.
Cette ressource fournit une opportunité unique pour explorer et comprendre les schémas d'utilisation des clusters informatiques à grande échelle, ainsi que pour développer des stratégies et des algorithmes efficaces pour gérer et optimiser de telles infrastructures.
Dans cette étude, nous nous pencherons sur l'analyse de la trace de Google afin de caractériser certains aspects de l'utilisation du cluster, en nous concentrant notamment sur l'identification des travaux et tâches les plus gourmands en CPU et en mémoire vive, ainsi que sur la classification des jobs par classe de priorité. Ce projet offre une opportunité précieuse de mettre en pratique les concepts et les outils d'analyse de données vus en cours, notamment Spark, tout en explorant des questions de recherche passionnantes dans le domaine du Big Data.

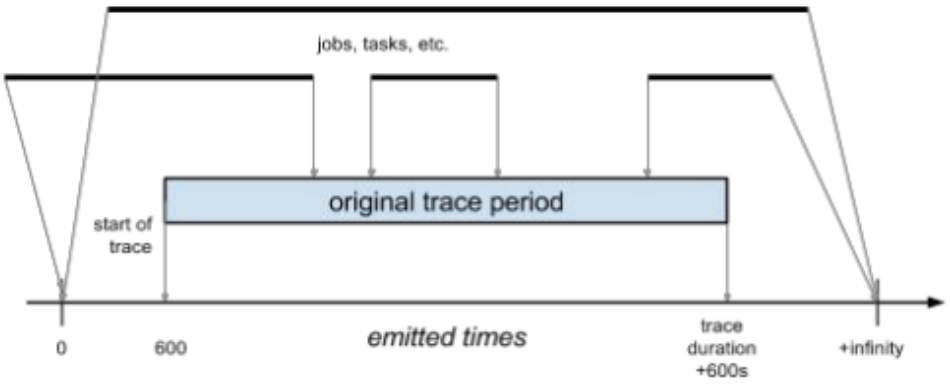
2 Méthodologie

2.1 Découverte de la trace

Le document débute par une présentation des clusters de Google, décrivant comment les machines sont organisées en racks et connectées via un réseau à haute bande passante. Il explique que les travaux informatiques (ou "jobs") sont les unités de travail principales, composées de "tâches" individuelles. Ces jobs et tâches sont gérés et programmés en fonction de leur cycle de vie et de leurs besoins en ressources.

- **Techniques de traitement des données :** Pour protéger la confidentialité, les données ont été modifiées. Par exemple, les textes libres sont hashés et les tailles de ressources sont transformées. Ces modifications visent à préserver l'utilité des données tout en protégeant les informations sensibles.
- **Tables de données :** Le document décrit diverses tables de données, telles que les événements des machines, les attributs des machines, les événements des jobs, etc. Chaque table capture des aspects spécifiques de l'utilisation du cluster, comme les changements d'état des machines ou les détails des tâches.

— **Gestion des jobs et des tâches :** Les jobs et les tâches passent par différents états (soumis, planifié, évincé, etc.). Chaque changement d'état est enregistré avec un type d'événement correspondant.



— **Usage des ressources :** L'utilisation des ressources est mesurée dans des conteneurs Linux, chaque tâche s'exécutant dans son propre conteneur. Les données couvrent l'utilisation du CPU, de la mémoire, de l'espace disque, et plus.

— **Format des fichiers :** Les données sont fournies en format CSV compressé. Les noms des fichiers suivent des modèles spécifiques indiquant la table de données à laquelle ils appartiennent.

— **Téléchargement des données :** Les traces sont stockées sur Google Storage for Developers et sont accessibles via des outils spécifiques ou via HTTP. Cela facilite l'accès aux données pour la recherche ou l'analyse.

— **Anomalies et informations manquantes :** Le document note des anomalies dans les traces, comme des informations manquantes ou des incohérences. Par exemple, certaines dépendances entre jobs ou d'autres types de ressources et contraintes ne sont pas représentées.

2.2 Choix du langage et des outils d'analyse

PySpark est un outil essentiel pour gérer efficacement le traitement de données à grande échelle. Lorsqu'il s'agit de traiter des ensembles de données massifs, les outils de traitement traditionnels sur une seule machine peuvent avoir du mal à faire face au volume, ce qui entraîne des pertes de performance et des temps de traitement longs.

PySpark, en revanche, exploite la puissance du calcul distribué en utilisant l'abstraction des ensembles de données distribués résilients (RDD) d'Apache Spark. En distribuant les données sur plusieurs nœuds dans un cluster et en parallélisant les calculs, PySpark peut traiter des quantités massives de données de manière

évolutive et tolérante aux pannes. Cette approche distribuée permet à PySpark de gérer les charges de travail de données volumineuses avec facilité, en offrant des temps de traitement plus rapides et en permettant aux organisations de tirer des résultats de leurs données de manière plus efficace.
De plus, PySpark fournit une API de haut niveau accessible via Python, ce qui nous permet de travailler avec des ensembles de données à grande échelle sans avoir besoin d'apprendre des frameworks de calcul distribué complexes. Dans l'ensemble, PySpark est un choix convaincant pour les tâches de traitement de données volumineuses, offrant une évolutivité, des performances et une facilité d'utilisation.

2.3 Analyse de la trace

- **Identification des jobs dominants en CPU** :
  Utiliser les données de task_usage pour identifier les jobs et tasks avec une utilisation maximale et moyenne du CPU (CPU rate, maximum CPU rate, sampled CPU usage). Trier et agréger ces données pour identifier les jobs les plus prenants en CPU.
- **Identification des jobs dominants en mémoire vive** :
  De même, utiliser les données de task_usage pour examiner l'utilisation de la mémoire (canonical memory usage, assigned memory usage, unmapped page cache, total page cache, maximum memory usage). Trier et agréger ces données pour identifier les jobs et tasks les plus prenants en mémoire vive.
- **Classification des jobs dominants par classe de priorité** :
  Combiner les données de task_events (en particulier les colonnes job ID et priority) avec les résultats des questions 1 et 2. Classer les jobs dominants identifiés précédemment selon leur priorité.
- **Étude de la corrélation entre la consommation de CPU et de la mémoire vive** :
  Utiliser les données de task_usage pour évaluer la corrélation entre CPU rate (ou des indicateurs similaires) et les différentes mesures de l'utilisation de la mémoire. Des analyses statistiques comme le calcul du coefficient de corrélation peuvent être appliquées ici.
- **Analyse de la durée des jobs et des tâches** :
  Étudier la distribution des durées des jobs et des tâches en utilisant les données de task_events et task_usage. Identifier les jobs et tâches qui durent plus longtemps que la moyenne, ainsi que ceux qui sont anormalement courts.

2.4 Nettoyage des données

Avant de procéder à l'analyse des données, un nettoyage préliminaire est nécessaire pour assurer la qualité et la fiabilité des données analysées. Un exemple de nettoyage des données est le filtrage des lignes où l'ID de job est NULL, ce qui élimine les données incomplètes ou non pertinentes pour l'analyse.

```
# Filtrer les lignes où l'ID de job est NULL
df_filtered = df.filter(~isnull("job ID"))
```

Ce processus de nettoyage est essentiel pour préparer les données à une analyse approfondie et garantir que les résultats soient basés sur des données complètes et précises.

2.5 Traitement des données

Le traitement des données implique la lecture, le filtrage, la sélection, et l'agrégation des données à partir des fichiers de trace pour effectuer les analyses décrites précédemment. Apache PySpark est utilisé pour gérer efficacement le volume important de données, en permettant des calculs distribués et en parallèle pour accélérer le processus d'analyse.

file pattern	field number	content	format	mandatory
job_events/part-????	1	time	INTEGER	YES
job_events/part-????	2	missing info	INTEGER	NO
job_events/part-????	3	job ID	INTEGER	YES
job_events/part-????	4	event type	INTEGER	YES
job_events/part-????	5	user	STRING_HASH	NO
job_events/part-????	6	scheduling class	INTEGER	NO
job_events/part-????	7	job name	STRING_HASH	NO
job_events/part-????	8	logical job name	STRING_HASH	NO
task_events/part-???'	1	time	INTEGER	YES

2.5.1 Exemple : Identification des tâches dominantes en utilisation du CPU

Pour identifier les tâches les plus gourmandes en CPU au sein d'un cluster informatique, nous avons employé Apache PySpark pour analyser et traiter un grand volume de données de trace. Le processus est décomposé en plusieurs étapes clés, comme décrit ci-après :

1. **Initialisation de SparkSession** : Une session Spark est initialisée pour permettre le traitement distribué des données. La configuration spécifie l'application comme étant dédiée à l'analyse des tâches dominantes en CPU et utilise tous les cœurs disponibles localement.

```
SparkSession.builder
    .appName("Top CPU Dominant Tasks")
    .master("local[*]")
    .getOrCreate()
```

2. **Préparation des chemins de fichiers** : Les chemins vers les fichiers de données sont préparés en utilisant les chemins d'accès et une liste de noms de fichiers, permettant de traiter chaque fichier de trace.
3. **Définition du schéma des données** : Un schéma est défini pour structurer les données lors de la lecture, comprenant des champs tels que l'ID du job, l'index de la tâche, et l'utilisation du CPU.
4. **Traitement des fichiers en boucle** : Chaque fichier de données est traité séquentiellement. Les données sont filtrées pour éliminer les entrées non pertinentes et sont ensuite agrégées pour calculer le taux moyen d'utilisation du CPU par tâche.

```
df_filtered.groupBy("job ID", "task index")
           .agg(avg("CPU rate").alias("average_cpu_rate"))
           .orderBy(desc("average_cpu_rate"))
           .limit(10)
```

5. **Accumulation des résultats intermédiaires** : Les résultats de chaque fichier sont accumulés pour identifier les tâches ayant le plus fort taux moyen d'utilisation du CPU à travers l'ensemble des données.
6. **Identification des tâches globalement dominantes** : Après avoir traité tous les fichiers, les données accumulées sont triées pour déterminer les 10 tâches les plus dominantes en termes d'utilisation du CPU.
7. **Affichage des résultats** : Les résultats finaux sont prêts à être affichés, montrant les tâches qui consomment le plus de CPU dans le cluster.

Cette méthode illustre l'utilisation efficace d'Apache Spark pour analyser de grandes quantités de données distribuées, permettant d'identifier les modèles d'utilisation des ressources dans un environnement de cluster.

3 Résultats

3.1 Identification des jobs dominants en CPU

job ID	task index	average_cpu_rate
6210686295	115	9.135018315864727
6419322512	10645	6.991486268786026
5115856683	12	4.14795548300026
4811385404	55	2.314707846955999
6295212302	67	1.6543118150597862
6295201188	78	1.3711346668501696
6000622471	16	1.2118503998654584
6210686295	72	1.2022168225958012
6000622471	2	1.1960032834322192

Analyser les tâches propres à chaque job nous a permis de mettre en évidence l'utilisation que la manière dont les jobs utilisent les performances du CPU. En observant les données de chaque job, nous avons pu identifier ceux ayant les valeurs les plus élevées, indiquant une forte demande en ressources de calcul.

Top des jobs par taux moyen d'utilisation du CPU :

— Identifiant du job **6210686295** : ce job est constitué de 3 tâches et présente un taux moyen d'utilisation du CPU d'environ 3,81 %. Il se distingue notamment avec la tâche ayant le taux d'utilisation du CPU le plus élevé : 9,14 %. De fait, ce job montre une demande significative en ressources CPU.

— Identifiant du job **5115856683** : Ce job, constitué d'une tâche unique, affiche le taux moyen d'utilisation du CPU le plus élevé, avec environ 4,15 %.

— Identifiant du job **6419322512** : Constitué de 2 tâches, ce job montre un taux moyen d'utilisation du CPU d'environ 3,76 %, avec un taux maximal d'utilisation du CPU par tâche de 6,99 %.

— Identifiant du job **4811385404** : Ce job à tâche unique présente un taux moyen d'utilisation du CPU de 2,31 %.

— Identifiant du job **6295212302** : Ce job à tâche unique présente un taux

moyen d'utilisation du CPU de 1,65 %.

Ces résultats mettent en lumière les jobs qui imposent la charge la plus significative sur les ressources CPU. De fait, les optimiser pourrait s'avérer crucial dans la planification des tâches et l'allocation des ressources.

3.2 Identification des jobs dominants en mémoire vive

job ID	task index	average canonical memory
2231855000000	2231908000000	6435122980.571428
1983324000000	1983325000000	6433233298.285714
2231861000000	2231880000000	6423984947.2
1966709000000	1966787000000	6418444902.4
1610315000000	1610338000000	6386025881.6
1878553000000	1878575000000	6384270208.0
1406162000000	1406314000000	6379294310.4
1007266000000	1007287000000	6345400661.333333
1888061000000	1888087000000	6343307946.666667

De façon similaire, l'examen de l'utilisation de la mémoire vive a révélé les tâches et jobs qui consomment le plus de RAM. Les valeurs obtenues pour chaque job montrent les processus qui ont besoin d'une grande quantité de mémoire, ce qui peut aider dans la prise de décisions pour la gestion des ressources.

Top des jobs par utilisation moyenne de la mémoire canonique :

L'ensemble des cinq jobs ci-dessous sont constitués d'une unique tâche.

— Identifiant du job **2231855000000** : Ce job affiche l'utilisation moyenne de la mémoire canonique la plus élevée, environ 6,44 GB.

— Identifiant du job **1983324000000** : Ce job affiche une utilisation moyenne de la mémoire canonique d'environ 6,43 GB.

— Identifiant du job **2231861000000** : Ce job a une utilisation moyenne de la mémoire canonique d'environ 6,42 GB.

— Identifiant du job **1966709000000** : Ce job possède une utilisation moyenne de la mémoire canonique d'environ 6,42 GB.

— Identifiant du job **1610315000000** : Celui-ci utilise en moyenne la mémoire canonique d'environ 6,39 GB.

Top des jobs par utilisation maximale de la mémoire canonique

— Identifiant du job **2404605000000** : Ce job est celui qui utilise le plus la capacité de la mémoire canonique avec environ 6,48 GB.

D'autre part, on retrouve dans ce classement, les jobs présents dans le top d'utilisation moyenne de la mémoire canonique. Cela indique de fait que ces jobs ont une consommation importante de RAM à la fois en moyenne et au pic maximal.

Avec ces résultats, on peut mettre en valeur les processus qui nécessitent d'importantes quantités de mémoire vive. Il s'agit d'une information cruciale pour allouer efficacement la mémoire.

3.3 Classification des jobs dominants par classe de priorité

job ID	task index	timestamp	missing info	machine ID	event type	user	scheduling class	priority
6272076905	0	163027215040	NULL	NULL	0	Gx2a4JlY7sTN3Jlqp	3	10
6272076905	0	1347760079184	NULL	4820073668	3	Gx2a4JlY7sTN3Jlqp	3	10
6272076905	0	163059602676	NULL	4820073668	1	Gx2a4JlY7sTN3Jlqp	3	10
6272076905	0	1347760079195	NULL	NULL	0	Gx2a4JlY7sTN3Jlqp	3	10
6272076905	0	1347790444482	NULL	257347123	1	Gx2a4JlY7sTN3Jlqp	3	10
6295212302	67	1976031581791	NULL	1334665910	5	tjYrEWcPX1rXEj2HG	2	9
6400512805	15	1738692084650	NULL	4802098790	5	oPxcKd7feXmw+sZK	0	9
6184860354	93	1555812792691	NULL	4821070690	5	y9NKAC+Ud/LMV2fx	2	9
4811385404	55	1613481994734	NULL	3889402292	5	rNyxTd1B3RnDJBIof	3	9
6184860354	273	1555813485370	NULL	4820235500	5	y9NKAC+Ud/LMV2fx	2	9

Combiner les données de consommation de CPU et de mémoire vive avec les niveaux de priorité des jobs, on nous a permis de révéler les informations suivantes pour les jobs ayant une consommation élevée de CPU :

— Identifiant du job **4811385404** : Avec une utilisation élevée du CPU, ce job est classé dans la classe de planification 3 avec une priorité de **9**.
— Identifiant du job **6295212302** : Avec une consommation significative de CPU, ce job est dans la classe de planification 2 et a également une priorité de **9**.
— Identifiant du job **6184860354** : Avec une utilisation moyenne du CPU moins élevée, ce job est classifié dans la classe de planification 2 avec une priorité de **9**.
— Identifiant du job **6272076905** : Avec une consommation de CPU notable, ce job est situé dans la classe de planification 3 avec la priorité la plus élevée de **10**.

— Identifiant du job **6400512805** : Avec une utilisation moyenne du CPU, ce job est classé dans la classe de planification 0 avec une priorité de **9**.

De fait, on remarque que la plupart des processus qui utilisent énormément de mémoire CPU sont considérés comme importants (prioritaires) par le système de gestion des jobs.

3.4 Étude de la corrélation entre la consommation de CPU et de la mémoire vive

correlation_cpu_canonical_memory	correlation_cpu_assigned_memory
0.352491425557883	0.002450362193947116

L'évaluation de la corrélation entre l'utilisation du CPU et celle de la mémoire vive révèle des fonctionnements différents selon le type de mémoire considéré. Les coefficients de corrélation fournissent un aperçu de la force et de la direction de ces relations :

- Corrélation entre l'utilisation du CPU et l'utilisation canonique de la mémoire : Le coefficient de corrélation est de 0.352491425557883, ce qui indique une corrélation modérée. Dans une certaine mesure, les jobs nécessitant davantage de ressources CPU tendent de fait à également utiliser plus de mémoire canonique.
Cette relation modérée peut être dû au fait que des tâches plus complexes ou exigeantes en calculs nécessitent d'utiliser simultanément la mémoire pour traiter les données.

- Corrélation entre l'utilisation du CPU et l'utilisation de la mémoire assignée : Le coefficient de corrélation est de 0.002450362193947116. Ici, la corrélation est quasi inexistante. En effet, l'augmentation de l'utilisation du CPU n'est donc pas liée à l'augmentation de l'utilisation de la mémoire assignée. Cela est sûrement dû au fait que la mémoire assignée est davantage liée à des politiques de gestion des ressources ou à des configurations préalables, plutôt qu'à la charge de travail réelle imposée par les tâches en cours d'exécution.

Ces résultats mettent en lumière que bien que l'utilisation du CPU et de la mémoire canonique partage une forme de relation, l'allocation de la mémoire semble suivre des critères indépendants de l'utilisation immédiate du CPU.
Cette distinction est importante pour comprendre comment optimiser les res-

sources dans des environnements de calcul intensif, en tenant compte non seulement de la demande en CPU mais aussi des besoins en mémoire des applications.

3.5 Analyse de la durée des jobs et des tâches

Statistique	Valeur
Nombre d'observations	123,279,930
Moyenne	253.65
Écart-type	100.16
Minimum	1.00
Maximum	300.00

TABLE 1 – Statistiques descriptives de la durée des tâches.

Nous avons examiné la répartition de la distribution des durées des jobs et des tâches afin d'identifier certaines anomalies et tendances. Les résultats montrent une large variabilité de la durée des jobs avec certains jobs durant bien plus longtemps que la moyenne tandis que d'autres sont beaucoup plus courts.

Compte : 123 279 930 observations de durée ont étés utilisées dans le calcul de la distribution.
Moyenne : La durée moyenne d'un job est d'environ 253.65 unités de temps.
Écart type : L'écart type est d'environ 100.16, ce qui indique une variation considérable dans la durée des jobs et tâches autour de la moyenne.
Minimum : La durée minimale observée est de 1.0 unité de temps, ce qui signifie que certains jobs ou tâches sont très brefs.
Maximum : La durée maximale est de 300.0 unités de temps, montrant que certains jobs ou tâches peuvent être extrêmement longs par rapport à la moyenne.

Ces résultats démontrent une large variabilité dans la durée des jobs et des tâches, avec une distribution qui inclut des durées extrêmement courtes mais aussi extrêmement longues. Cela peut s'expliquer par l'existence de plusieurs types de tâches, certaines nécessitant un traitement rapide et d'autres de longs processus de calcul.

4 Conclusion

À travers les analyses, il apparaît comme évident que certaines tâches monopolisent les ressources et pourraient bénéficier d'une optimisation. La classification

des jobs en fonction de leur priorité a permis de révéler que les jobs à haute priorité ne sont pas nécessairement ceux qui utilisent le plus de ressources.
Cette découverte pourrait être exploitée pour ajuster la planification des tâches et l'allocation des ressources. En effet, revoir le fonctionnement du système de gestion des priorités pour les jobs pourrait permettre à Google d'optimiser la gestion de ses ressources.