INTERNATIONAL DOCTORAL SCHOOL OF THE USC

Pablo
Vázquez Caderno

PhD Thesis

BigOPERA: An OPportunistic and Elastic Resource Allocation for Big Data Frameworks

Santiago de Compostela, 2025

Doctoral Programme in Information Technology Research

ESCOLA DE DOUTORAMENTO
INTERNACIONAL DA USC

DOCTORAL THESIS

BIGOPERA: AN OPPORTUNISTIC AND ELASTIC RESOURCE ALLOCATION FOR BIG DATA FRAMEWORKS

Author

Pablo Vázquez Caderno

Supervisors: José Carlos Cabaleiro Domínguez

Feras Mahmoud Naji Awaysheh

PHD PROGRAMME IN NVESTIGACIÓN EN TECNOLOXÍAS DA INFORMACIÓN.

SANTIAGO DE COMPOSTELA

Acknowledgments

I would like to thank and dedicate this work to all the people who, in one way or another, contributed to it. Paraphrasing Newton “If I have finished this thesis, it is by standing on the shoulders of giants". I am deeply grateful to my supervisors for their guidance and for allowing me to pursue this work. Speaking of journeys, I want to thank Lau, my companion on this endless journey and the love of my life, for her encouragement and help so I could work on this project. To Alteia, since your arrival, the Sun shines brighter. You both make every day worth living. Por último, quero agradecer aos meus pais, amigos e a toda a xente que fixo posible que un neno de aldea coma Balbino, puidese estudar na universidade.

A vós.

27 de maio de 2025

Abstract

The rapid growth of data-intensive applications has driven the need for more scalable, flexible, and sustainable resource management in Big Data (BD) frameworks. Traditional computing infrastructures often rely exclusively on dedicated resources, which can lead to inefficient utilization and increased operational costs. To address this challenge, this thesis explores the integration of opportunistic computing into Apache Spark through a hybrid resource allocation framework named BigOPERA. BigOPERA combines the elasticity of opportunistic nodes, machines not primarily dedicated to the cluster, with the stability of dedicated infrastructure, achieving cost-aware scalability without compromising performance. The system architecture integrates Apache Spark in standalone mode for primary data processing, Docker for containerized task isolation, and HTCondor as the orchestrator for opportunistic resource provisioning. Modifications to Spark internal scheduling logic allow for dynamic executor placement that prioritizes dedicated nodes and leverages opportunistic ones as fallback options. The BigOPERA framework chapter details the design rationale, including using containerized workers and implementing a two-tier scheduling algorithm integrating dedicated and opportunistic resources. It also discusses the networking setup and deployment flexibility required to support hybrid cluster configurations. The framework is evaluated through synthetic benchmarks (HiBench) and a real-world genomics use case involving UMI-based single-cell RNA sequencing. This included comparing execution times for sorting and UMI counting operations against native Spark and High-Performance Computing (HPC) baselines, using varying data sizes to assess scalability. Experiments were conducted under various workloads on Google Cloud Platform, focusing on performance and fault tolerance across dedicated and hybrid cluster configurations. Results demonstrate that BigOPERA outperforms standard Spark deployments by up to 30% in execution time for specific workloads while reducing idle time and infrastructure costs. The hybrid scheduling algorithm shows advantages for shuffle-intensive and iterative tasks despite diminishing returns at large data sizes due to resource saturation and opportunistic instance limitations. This thesis contributes a novel, production-ready approach to extending Spark capabilities with opportunistic computing. It demonstrates how standard Spark workloads can obtain performance gains without requiring user application modifications, while at the same time presenting a two-tiered scheduler that efficiently manages opportunistic resource allocation. Beyond performance improvements, the proposed framework increases fault tolerance by distributing workloads across a more diverse and dynamic set of resources, thereby reducing reliance on any single infrastructure component. Additionally, it contributes to reducing the carbon footprint of

data centres by utilizing otherwise idle resources, aligning with sustainable computing practices, highlighting the environmental relevance of this approach.

BigOPERA opens new possibilities for elastic Big Data analytics and paves the way for more adaptable and energy-aware cluster architectures.

Contents

Resumo

Nesta tese abórdase un dos retos fundamentais no ámbito da computación distribuída moderna: a baixa eficiencia estrutural na utilización dos recursos dispoñibles, particularmente en escenarios nos que se manexan grandes volumes de datos. A pesar dos avances significativos en frameworks como Apache Spark, a maior parte das infraestruturas de procesamento masivo continúan a depender de modelos baseados exclusivamente en recursos dedicados. Este enfoque garante certa estabilidade operativa, mais presenta importantes limitacións tanto no rendemento como no custo enerxético e económico. Co obxectivo de paliar estas ineficiencias, propóñese a arquitectura BigOPERA, un sistema híbrido deseñado para integrar de forma dinámica recursos dedicados e oportunistas. Esta integración permite ampliar elasticamente a capacidade dispoñible en tempo de execución, sen modificar a lóxica das aplicacións existentes nin comprometer a tolerancia a fallos. BigOPERA dota a Apache Spark dunha nova capa de planificación que fai posible o aproveitamento de recursos efémeros, mellorando así a eficiencia global do sistema de forma transparente para o usuario final.

O contexto tecnolóxico no que se enmarca esta investigación está caracterizado por un crecemento exponencial e sostido na xeración de datos a escala global. A prolífica expansión de dispositivos móbiles, sensores industriais, sistemas de videovixilancia, instrumentos científicos de alto rendemento, como secuenciadores de nova xeración ou radiotelescopios, está a xerar unha cantidade de información sen precedentes. Esta situación require non só capacidade de almacenamento masivo, senón tamén solucións eficaces que permitan procesar e analizar os datos en tempo útil, respondendo a necesidades operativas, científicas ou empresariais con garantías de escalabilidade e fiabilidade.

Historicamente, a resposta a esta demanda materializouse na implantación de clústers dedicados, configurados para estar permanentemente dispoñibles e atender cargas de traballo intensivas. Con todo, este modelo tradicional presenta importantes limitacións. Por unha banda,

obriga a sobredimensionar a infraestrutura para cubrir os picos de demanda, provocando unha situación habitual de infrautilización na maior parte do tempo. Por outra, a dispoñibilidade constante destes recursos implica un consumo enerxético elevado e continuo, mesmo en momentos de baixa actividade, incrementando o custo económico das operacións e agravando a súa pegada ambiental.

Segundo estimacións recentes da International Energy Agency, os centros de datos superan xa os 200 TWh de consumo eléctrico anual, unha cifra equiparable á demanda total de países medianos. Aínda que existen esforzos técnicos dirixidos a mellorar a eficiencia de compoñentes individuais como procesadores, dispositivos de almacenamento ou sistemas de refrixeración, a arquitectura xeral de planificación e asignación de recursos continúa a ser, en termos globais, altamente ineficiente. Nun contexto marcado pola emerxencia climática e as políticas de transición enerxética, resulta indispensable revisar os paradigmas actuais de computación intensiva e explorar modelos máis flexibles e sostibles.

Simultaneamente, moitas organizacións, centros de investigación e entidades públicas contan con volumes considerables de recursos infrautilizados. Equipos de escritorio, servidores en desuso, clústers secundarios ou instancias cloud reservadas para proxectos puntuais permanecen inactivos boa parte do tempo. O aproveitamento destes recursos, sen interferir co seu uso principal, constitúe a esencia da computación oportunista. Lonxe de ser unha proposta recente, este paradigma ten raíces sólidas en iniciativas pioneiras como o proxecto Condor, desenvolvido a finais da década de 1980, que permitía empregar ciclos ociosos de computadoras persoais distribuídas en rede.

A converxencia entre estas dúas liñas, a procura dunha maior eficiencia na escalabilidade de sistemas Big Data e a valorización dos recursos computacionais inactivos, serve de punto de partida para esta tese. A proposta formulada consiste en reintroducir o modelo de computación oportunista no contexto actual, integrándoo directamente en Apache Spark e ofrecendo así unha solución moderna, dinámica e transparente que permita a expansión da infraestrutura de execución cando as condicións o permitan, sen comprometer a robustez nin a reproducibilidade das tarefas programadas.

O propósito central que orienta esta investigación é o deseño, implementación e avaliación dunha arquitectura híbrida que permita incorporar, de maneira transparente e eficiente, recursos computacionais non dedicados no marco do ecosistema Apache Spark. A proposta, denominada BigOPERA, pretende estender a lóxica de planificación tradicional de Spark mediante a integración de recursos efémeros, tamén chamados oportunistas, ofrecendo así

unha alternativa máis elástica, tolerante a fallos e sensible ao consumo enerxético para o procesamento distribuído de grandes volumes de datos.

Para acadar este obxectivo xeral, a tese estrutúrase en torno a varias metas técnicas e experimentais específicas. En primeiro lugar, abórdase o deseño dun modelo de planificación que permita distinguir explicitamente entre recursos dedicados, dispoñibles de forma estable, e recursos oportunistas, dispoñibles de maneira intermitente e sen garantías de continuidade. Este modelo debe ser capaz de preservar a estabilidade do sistema base, priorizando os nodos dedicados cando sexa necesario, pero tamén debe estar preparado para aproveitar a capacidade adicional dispoñible cando esta emerxa, sen provocar penalizacións inaceptables nin comprometer a integridade da execución.

En segundo lugar, proponse a integración dun sistema externo de xestión de recursos, neste caso HTCondor, co ecosistema de Spark, configurando un mecanismo de coordinación baseado en contedores e scripts específicos que permitan xestionar o ciclo de vida dos executores de forma autónoma. Este enfoque permite lanzar tarefas en nodos efémeros sen modificar o comportamento do código do usuario, garantindo ao mesmo tempo a compatibilidade cos compoñentes internos de Spark.

Outro dos piares da proposta consiste na adaptación do planificador interno de Spark para que poida recoñecer e tratar de maneira diferenciada os distintos tipos de nodos. Isto implica modificar a lóxica de asignación de tarefas, incluíndo políticas que contemplen a natureza efémera dalgúns recursos e permitan reprogramar de forma automática as tarefas fallidas derivadas da súa perda. O obxectivo é garantir unha execución fluída, mesmo en contornos cunha elevada variabilidade na dispoñibilidade dos recursos.

Así mesmo, establécese un modelo empírico de consumo enerxético aplicable aos servidores empregados, co obxectivo de identificar os factores clave que condicionan a eficiencia segundo o tipo de carga procesada, o número de núcleos activos e a frecuencia do procesador. Esta análise permite comprender en que condicións se acada unha mellor relación entre rendemento e consumo, servindo de base para o deseño de políticas de planificación máis eficientes.

Finalmente, a proposta é avaliada de maneira sistemática mediante o uso de benchmarks recoñecidos como Intel HiBench e casos de uso representativos, co obxectivo de cuantificar o impacto real da arquitectura en termos de tempo de execución, tolerancia a fallos, escalabilidade horizontal e consumo enerxético. A análise dos resultados obtense baixo diferentes configuracións e escenarios experimentais, e serve como base para propoñer futuras liñas de

traballo. Entre estas destacan a integración de métricas enerxéticas en tempo de execución como criterio de planificación, así como o desenvolvemento de modelos predictivos que permitan anticipar a dispoñibilidade dos recursos e mellorar a asignación dinámica das tarefas en contornos reais.

A sección dedicada aos traballos relacionados dentro do Capítulo 2 recolle unha revisión crítica da literatura centrada en tres liñas principais: (i) frameworks e sistemas de execución para o procesamento distribuído orientado ao Big Data, (ii) arquitecturas de xestión de recursos con capacidades de elasticidade e integración dinámica, e (iii) enfoques relacionados coa computación oportunista en contornos heteroxéneos. Esta análise permite situar a proposta de BigOPERA no contexto das iniciativas existentes, identificando as súas achegas específicas e as lagoas que pretende cubrir en relación á integración de recursos non dedicados.

No primeiro dos eixes, revísanse plataformas como Mesos e Kubernetes, deseñadas para compartir infraestrutura entre múltiples frameworks de execución. Mesos propón unha arquitectura de compartición de clústers baseada nunha asignación fina de recursos entre aplicacións concorrentes. Pola súa banda, Kubernetes consolidouse como o estándar de facto para a orquestración de contedores en contornos cloud-native, destacando pola súa elasticidade e tolerancia a fallos. Non obstante, estas solucións adoitan operar en contornos controlados e non contemplan mecanismos internos para a xestión directa de nodos efémeros por parte do planificador de frameworks como Spark. A súa elasticidade baséase na orquestración externa, o que dificulta a súa integración transparente con sistemas de execución que dependen de estado interno.

No segundo eixe, analízase a evolución de Apache Spark como o framework dominante para o procesamento de datos en memoria a gran escala. A súa arquitectura driver-executor, combinada co modelo de RDDs e a compatibilidade con linguaxes como Scala e Python, converteuno nun estándar de facto. Con todo, Spark foi deseñado asumindo estabilidade na dispoñibilidade dos recursos. Este deseño limita a súa capacidade para funcionar en contornos onde a dispoñibilidade de nodos pode variar dinamicamente. Neste contexto, destaca a proposta de Sparrow, un planificador distribuído deseñado para workloads de baixa latencia, que emprega técnicas como late binding e batch sampling para acadar decisións rápidas e eficientes en clusters con alta concorrencia. Aínda que non está orientado á computación oportunista, Sparrow representa unha liña de investigación relevante na evolución cara a sistemas de planificación máis flexibles, que tamén inflúe indirectamente nas motivacións de BigOPERA.

Por último, no terceiro eixe, abórdase o ámbito da computación oportunista, representado

por sistemas como HTCondor, así como propostas recentes centradas en Spark. TR-Spark explora a tolerancia a fallos mediante réplicas de tarefas, mentres que OpERA modifica o planificador de Hadoop YARN para mellorar o uso dos recursos en función da carga en tempo de execución. MOON, pola súa banda, explora o uso de recursos oportunistas para workloads MapReduce, destacando a importancia da adaptabilidade fronte á variabilidade dos recursos dispoñibles. Estas achegas comparten co presente traballo a intención de mellorar a utilización de recursos e reducir os tempos de execución en contornos Big Data. Con todo, tanto TR-Spark como OpERA operan sobre recursos dedicados e non abordan directamente a integración de recursos non garantidos. Ademais, en contornos cloud, tamén se estudaron mecanismos para aproveitar instancias spot e reducir custos sen perder rendemento, mais sen ofrecer un modelo xeral aplicable tamén a contornos locais.

A análise destes tres eixes evidencia que non existe, ata o de agora, unha solución que permita a integración efectiva, transparente e tolerante a fallos de recursos non dedicados en Apache Spark sen alterar o seu núcleo nin os algoritmos dos usuarios. Esta constatación serve de base para o desenvolvemento de BigOPERA, unha arquitectura híbrida que introduce mecanismos para incorporar recursos oportunistas ao ecosistema Spark, mantendo a compatibilidade co modelo existente e permitindo escalar a infraestrutura dispoñible segundo a presenza real de nodos voluntarios ou efémeros no sistema.

A arquitectura proposta en BigOPERA, descrita en detalle no Capítulo 3, foi concibida coa finalidade de posibilitar a integración efectiva entre dous sistemas que responden a paradigmas moi distintos de xestión e execución de tarefas computacionais. Por unha banda, Apache Spark, deseñado para operar sobre recursos dedicados en contornos estables e controlados, actúa como motor de execución principal; por outra, HTCondor, cunha longa traxectoria na xestión de recursos intermitentes e heteroxéneos, proporciona a infraestrutura para descubrir e asignar recursos oportunistas de forma dinámica. A integración entre ambos sistemas articúlase arredor dun deseño modular, no que cada compoñente mantén a súa lóxica independente, pero colabora de forma coordinada para ofrecer unha infraestrutura elástica, tolerante a fallos e adaptada á realidade dos recursos dispoñibles.

A arquitectura estrutúrase en tres capas funcionais ben definidas. A primeira delas, correspondente á xestión de recursos, delega en HTCondor a responsabilidade de identificar nodos oportunistas dispoñibles, reservar a súa capacidade e lanzalos cando se cumpren as condicións definidas polas políticas de uso. Cada nodo candidato executa os compoñentes necesarios para anunciar a súa dispoñibilidade ao sistema central de coordinación e recibir tarefas cando é

seleccionado para participar na execución.

A segunda capa, centrada na planificación e execución, reside en Spark, onde o `driver` toma as decisións sobre a asignación de executores en función da dispoñibilidade e da natureza dos nodos. A terceira capa, de integración, encapsula os scripts de lanzamento e os contedores Docker que permiten instanciar executores de Spark en nodos asignados por HTCondor, facilitando a súa rexistración dinámica e sen intervención manual no proceso de execución global.

O funcionamento conxunto destas capas desenvólvese segundo un fluxo claramente definido: cando un nodo se detecta como dispoñible, HTCondor programa un traballo que consiste nun script encargado de lanzar un contedor con Apache Spark preconfigurado. Unha vez iniciado, o contedor executa o binario `spark-class` co rol de executor, conectándose ao `driver` especificado e rexistrándose para recibir tarefas. No caso de que o nodo sexa reclamado polo seu usuario principal, HTCondor remata a tarefa e interrompe o contedor. Spark detecta automaticamente a perda do executor e reprograma as tarefas afectadas segundo a súa lóxica de tolerancia a fallos. Este comportamento garante que Spark poida operar sen necesidade de distinguir explicitamente entre nodos dedicados e efémeros, asumindo como válidos todos os eventos que afecten á dispoñibilidade de executores.

Desde o punto de vista da portabilidade e reproducibilidade, a arquitectura apoia o uso de contedores Docker que encapsulan todos os compoñentes necesarios para a execución: unha versión oficial de Spark (v3.2 ou superior), scripts parametrizables de lanzamento que interpretan as variables de entorno proporcionadas por HTCondor, e mecanismos de rexistro e monitorización para facilitar a supervisión da execución. Estes contedores, ao seren autocoherentes e facilmente despregables, permiten replicar a configuración en diferentes clústers, mesmo con perfís de hardware heteroxéneos.

Para xestionar o ciclo de vida dos nodos e anticiparse a eventos de perda ou preempción, defínese un modelo de estados inspirado na semántica de HTCondor. Este modelo inclúe os estados Unclaimed, Matched, Claimed, Owner, Preempting e Error, permitindo que o sistema reaccione de forma eficaz ante cambios na dispoñibilidade dos recursos e que adopte medidas preventivas de reprogramación cando sexa necesario.

No que respecta á planificación interna de Spark, realizouse unha adaptación da función `scheduleExecutorsOnWorkers`, encargada de decidir a asignación de tarefas a executores dispoñibles. Esta modificación permite distinguir entre nodos dedicados e oportunistas, procesando os primeiros con prioridade. Ademais, a opción por defecto de `spreadOutApps`

permite dispersar as tarefas entre varios nodos, reducindo a vulnerabilidade ante fallos masivos. Esta combinación de estratexias garante un comportamento robusto, capaz de adaptarse á dispoñibilidade dinámica dos recursos sen comprometer a execución global.

A natureza efémera dos recursos oportunistas representa un dos desafíos máis críticos no deseño de arquitecturas híbridas tolerantes a fallos. En contextos reais, estes recursos poden ser recuperados polo seu propietario, apagados inesperadamente ou quedar fóra de servizo por problemas de conectividade. Esta inestabilidade introduce un grao de imprevisibilidade que resulta particularmente problemático en frameworks como Apache Spark, cuxa arquitectura inicial non estaba deseñada para xestionar a perda frecuente de executores. De feito, estudos previos sinalaron que versións temperás de Spark sufrían fallos sistemáticos ao intentar completar tarefas en contornos con recursos volátiles, conducindo a erros de execución difíciles de diagnosticar e resolver.

Co obxectivo de avaliar a capacidade de BigOPERA para operar baixo estas condicións adversas, deseñouse unha proba baseada na execución do benchmark `SparkPi` con 800.000 particións, replicando o escenario do apartado 4.1 pero introducindo interrupcións simuladas nos nodos oportunistas. As desconexións foron realizadas de forma aleatoria, asumindo independencia entre eventos, mediante scripts que detiñan e reiniciaban os procesos xestionados por HTCondor. Os resultados amosaron que, malia a variabilidade esperada nos tempos de execución, o clúster híbrido foi consistentemente máis rápido que o clúster dedicado, o que demostra que a utilización eficiente de recursos ociosos pode compensar as interrupcións ocasionais. Estes achados permiten concluír que BigOPERA, grazas á súa integración de mecanismos de detección automática de fallos, reprogramación de tarefas e uso de almacenamento distribuído tolerante a erros como HDFS, ofrece unha solución robusta que permite completar aplicacións incluso en escenarios con recursos inestables. A capacidade do sistema para adaptarse dinámicamente ás condicións do entorno confirma a súa viabilidade como plataforma resiliente para computación distribuída en contornos heteroxéneos e non deterministas.

En síntese, BigOPERA presenta unha solución que mantén compatibilidade total co ecosistema Spark, sen requirir modificacións nos algoritmos dos usuarios nin nos mecanismos estándar de execución. A súa arquitectura modular, o uso de contedores e a integración transparente con HTCondor fan posible o despregue da plataforma en contornos variados, tanto locais como híbridos ou federados, ofrecendo unha vía realista para aproveitar recursos infrautilizados e mellorar a eficiencia da computación distribuída.

A validación experimental da arquitectura BigOPERA constitúe un dos elementos centrais

desta investigación, e está detallada no Capítulo 4. Para levar a cabo esta avaliación empregouse como ferramenta principal o conxunto de benchmarks Intel HiBench, amplamente recoñecido pola súa representatividade e cobertura de workloads comúns no ámbito do Big Data. Este marco de probas inclúe tarefas diversas que permiten simular escenarios de computación intensiva, procesamento de texto a gran escala, operacións de clasificación e algoritmos de agrupamento. A súa diversidade e fidelidade ao comportamento real de aplicacións distribuídas converten a Intel HiBench nun instrumento particularmente axeitado para medir o rendemento, a escalabilidade e a robustez de arquitecturas como a proposta.

Os experimentos foron deseñados co obxectivo de analizar o comportamento de BigOPERA baixo distintos perfís de infraestrutura. Para iso, compararonse dúas configuracións principais: un clúster composto exclusivamente por nodos dedicados, empregado como liña base estática e unha configuración híbrida que integra o clúster dedicado con nodos oportunistas, xestionados por HTCondor, que permite observar o impacto da computación oportunista. Esta última representa o enfoque adoptado por BigOPERA, e permite avaliar o sistema en condicións máis realistas e dinámicas. Ademais, introducíronse taxas de perda simuladas para cuantificar a capacidade do sistema para adaptarse a escenarios de alta volatilidade. A métrica analizada foi o tempo total de execución por tarefa ou workload.

Os resultados obtidos ao longo destas probas ofrecen evidencias consistentes do valor da arquitectura proposta. En todos os workloads avaliados: `Sort`, `WordCount`, `TeraSort` e `SparkPi`, a configuración híbrida asociada a BigOPERA demostrou ser máis eficiente que a configuración puramente dedicada entre un 10 e un 35 %. Estes datos amosan que a arquitectura é capaz de aproveitar de forma eficaz os recursos efémeros dispoñibles sen comprometer a continuidade da execución. A capacidade do sistema para reprogramar tarefas automaticamente, sen intervención manual e sen necesidade de reinicio global, constitúe unha das súas principais fortalezas. Esta resiliencia, combinada cunha planificación consciente da natureza dos recursos, permite que a execución se adapte en tempo real aos cambios na infraestrutura dispoñible.

Cómpre salientar, ademais, o efecto positivo da activación da opción `spreadOutApps` a `true`, que promove unha dispersión estratéxica das tarefas entre distintos nodos do clúster. Esta opción é ser particularmente eficaz para mitigar o impacto de fallos catastróficos, ao evitar a concentración de tarefas críticas en executores illados. A través desta estratexia, o sistema logra manter un equilibrio máis robusto, distribuíndo a carga de forma que se minimicen os riscos asociados á perda simultánea de varios compoñentes. Esta capacidade de adaptación e

balanceo dinámico reflicte un deseño consciente da realidade dos contornos heteroxéneos e non garantidos, nos que a dispoñibilidade dos recursos pode cambiar de maneira imprevisible.

En conxunto, os achados experimentais confirman a viabilidade técnica e práctica de BigOPERA como unha solución escalable, eficiente e resiliente. A súa capacidade para integrar recursos oportunistas sen comprometer o comportamento global da aplicación posiciona esta arquitectura como unha alternativa sólida para o procesamento distribuído en escenarios nos que a elasticidade, o custo e a eficiencia enerxética constitúen factores determinantes.

Co obxectivo de validar a aplicabilidade de BigOPERA en escenarios computacionais reais e con elevados requirimentos de procesamento, desenvolveuse un caso práctico baseado na implementación do pipeline `OPERA-gSAM`, descrito no Capítulo 5. Este fluxo de traballo foi especificamente deseñado para o tratamento de datos xenómicos procedentes de experimentos de ARN de célula única (scRNA-seq), unha modalidade analítica que implica un elevado volume de datos semiestruturados e que esixe un uso intensivo da CPU, acceso aleatorio ao disco e unha alta capacidade de xestión de memoria. A cadea de procesamento inclúe diversas fases fundamentais, como a lectura de ficheiros `SAM`, o filtrado de lecturas segundo métricas de calidade, a ordenación das lecturas atendendo ás súas coordenadas xenómicas, a identificación e reconto de moléculas únicas mediante identificadores UMI, e a conversión final dos resultados a formato `Parquet`, que permite unha representación comprimida e optimizada para consultas posteriores.

A execución deste pipeline realizouse tanto en contornos Spark convencionais compostos exclusivamente por nodos dedicados como en contornos mixtos xestionados por BigOPERA, nos que se combinan recursos dedicados e oportunistas. Os resultados obtidos permiten observar melloras substanciais en varios parámetros de interese. En primeiro lugar, o uso do formato `Parquet` permitiu reducir significativamente o espazo de almacenamento requirido, con aforros de ata un 60 por cento respecto de formatos tradicionais como CSV ou SAM. Isto supón unha vantaxe considerable en termos de eficiencia de disco e custo de almacenamento. En segundo lugar, as medicións de rendemento amosan que, grazas á capacidade de BigOPERA para incorporar recursos adicionais de forma dinámica, os tempos totais de execución reducíronse entre un 25 e un 40 por cento, dependendo da proporción e estabilidade dos nodos oportunistas empregados. Este comportamento demostra que a arquitectura proposta é capaz de adaptarse eficientemente ás flutuacións na dispoñibilidade dos recursos sen comprometer a integridade do procesamento.

En conxunto, esta validación empírica confirma que BigOPERA é unha arquitectura apta

para ser utilizada en tarefas reais de alta demanda computacional, nas que a combinación de eficiencia, flexibilidade e tolerancia a fallos resulta determinante para o éxito das operacións.

As conclusións desta tese evidencian que é tecnicamente viable e beneficioso integrar computación oportunista en frameworks modernos de procesamento distribuído como Apache Spark. A arquitectura BigOPERA permite un aproveitamento máis eficiente da infraestrutura dispoñible ao ampliar dinámicamente os recursos computacionais sen necesidade de intervención manual e sen comprometer a compatibilidade co núcleo funcional de Spark. A solución proposta demostra unha elevada robustez fronte a escenarios con nodos efémeros ou voláteis, mantendo a estabilidade das execucións incluso en condicións de perda de recursos. Ademais, ao non requirir modificacións nos algoritmos dos usuarios nin na configuración dos seus programas, BigOPERA resulta facilmente adoptable tanto en contornos locais como en infraestruturas híbridas ou nativas da nube.

De cara ao futuro, identifícanse diversas liñas de traballo orientadas a ampliar o alcance e mellorar a eficiencia da arquitectura desenvolvida. Un primeiro obxectivo é continuar refinando os algoritmos de integración e planificación para adaptalos a contornos computacionais aínda máis heteroxéneos, explorando a súa aplicación a outros frameworks de Big Data alén de Spark e estendendo a súa utilidade a escenarios de procesamento de datos en tempo real. Neste sentido, a incorporación de técnicas avanzadas de aprendizaxe automática para a predición da dispoñibilidade de recursos emerxe como unha vía prometedora para mellorar tanto o rendemento como a asignación óptima de tarefas. A automatización da planificación baseada en modelos predictivos permitiría unha resposta máis eficiente ante flutuacións dinámicas na infraestrutura dispoñible.

Outra liña de investigación clave consiste na extensión do modelo actual, que opera a nivel de asignación de executores, cara a un enfoque máis granular baseado na adaptación do propio TaskScheduler de Spark. Esta capa de planificación, responsable de distribuír tarefas entre executores activos, ofrece unha oportunidade única para aplicar decisións máis afinadas, como asignar tarefas críticas ou sensibles á latencia a nodos dedicados, mentres que tarefas menos prioritarias ou especulativas poderían ser dirixidas a nodos oportunistas. Esta evolución permitiría a BigOPERA responder de forma máis intelixente a condicións de execución en tempo real e recuperar tarefas con maior eficacia en presenza de perdas de nodos, integrando a lóxica de planificación oportunista no núcleo mesmo da execución distribuída.

Para validar esta proposta, resulta esencial implementar un prototipo de TaskScheduler con soporte para nodos oportunistas e comparalo empiricamente co mecanismo actual de

planificación a nivel de executor. Esta análise permitiría avaliar diferenzas en métricas como o throughput global, a latencia media por tarefa, a utilización efectiva dos recursos e a capacidade de recuperación do sistema. Un resultado positivo nesta comparación confirmaría que os beneficios da planificación oportunista poden estenderse máis alá da asignación inicial de recursos, abrindo a porta a unha coordinación adaptativa a nivel de tarefa.

Desde a perspectiva da sustentabilidade, outra liña fundamental de desenvolvemento refírese ao consumo enerxético asociado ao uso de recursos oportunistas. Os experimentos desta tese demostraron que a utilización intensiva de todos os recursos dispoñibles non sempre resulta na opción máis eficiente en termos de enerxía. Estudos previos indican que o consumo da CPU non escala de forma lineal coa carga e que a eficiencia máxima adoita alcanzarse con frecuencias moderadas e todos os núcleos activos, en lugar de operar ao máximo rendemento continuo. Por este motivo, propóñese como futuro traballo a análise sistemática de distintas configuracións de HTCondor, co obxectivo de identificar os parámetros que permiten equilibrar rendemento e consumo enerxético, así como establecer políticas de activación de recursos que teñan en conta a eficiencia térmica e eléctrica real dos nodos dispoñibles.

Neste marco, tamén se propón estender o modelo de HTCondor para incorporar explicitamente a eficiencia enerxética como criterio de planificación. Isto podería materializarse mediante a definición dun atributo personalizado para cada nodo, baseado en medicións empíricas de consumo por unidade de traballo computacional. Esta información permitiría ao planificador priorizar nodos cunha mellor relación rendemento-vatio, mellorando así a eficiencia global do sistema sen renunciar á súa capacidade de adaptación. A modificación da lóxica de matchmaking e ranking permitiría unha asignación máis intelixente das tarefas, favorecendo aquelas configuracións que minimicen o desperdicio enerxético sen sacrificar throughput.

En resumo, BigOPERA abre novas posibilidades para o deseño de arquitecturas distribuídas que combinen elasticidade, eficiencia e sustentabilidade. As liñas de traballo identificadas permitirán consolidar o sistema como unha solución pioneira na xestión eficiente de recursos en contornos de Big Data, ofrecendo mecanismos de planificación adaptativos, enerxeticamente conscientes e aplicables a un amplo abano de escenarios computacionais presentes e futuros.

Chapter 1

Introduction

The rapid expansion of digital data has fundamentally changed the way organizations process information, make decisions, and generate value. Big Data (BD), defined by characteristics such as volume, velocity, variety, and veracity, has positioned data analytics as a central component of contemporary business practices and scientific inquiry [1].

The volume of data is enormous, measured in terabytes, petabytes, and even exabytes, and continues to grow exponentially. Velocity speaks to the speed at which data are generated, processed, and transmitted, often requiring real-time or near-real-time analysis. Variety reflects the diverse nature of data sources and formats, encompassing numerical data, textual content, multimedia, and more. Veracity underscores the importance of data quality, accuracy, and trustworthiness. These dimensions create a dynamic and complex data ecosystem that defies conventional data management and analysis approaches.

Big Data analytics represents a transformative paradigm that applies advanced techniques, algorithms, and technologies to extract patterns, trends, correlations, and meaningful insights from vast and complex datasets. It is a multidisciplinary field that integrates principles from data science, statistics, computer science, and domain-specific knowledge to uncover actionable intelligence. By leveraging these insights, organizations and individuals can make informed decisions, run their operations more efficiently, better understand and serve their customers, and develop new products or services.

The power of Big Data analytics lies in its ability to harness the immense and diverse data landscape, which is both a challenge and an imperative in today's data-driven world [2]. It has become a cornerstone for organizations seeking to uncover hidden opportunities, fos-

ter innovation, and maintain industry competitiveness. Its applications span across domains such as finance, healthcare, marketing, manufacturing, and scientific research, where it supports efficient data processing and helps tackle domain-specific analytical and computational challenges. Its applications span across domains such as finance, healthcare, marketing, manufacturing, and scientific research, where it supports efficient data processing and helps tackle domain-specific analytical and computational challenges.

Today, industry and academia are increasingly powered by Big Data, and their operations are increasingly focused on accommodating the demands of high-performance and high-throughput applications [3]. This shift underscores the critical role of Big Data analytics in overcoming challenges across multiple disciplines and driving progress in an increasingly data-centric era. Among these disciplines, genomics stands out as a field generating massive volumes of structured and semi-structured data through technologies such as Next-Generation Sequencing (NGS) and single-cell RNA sequencing (scRNA-seq). Processing such data requires intensive computation for alignment, sorting, and quantifying reads, often involving complex, multi-stage pipelines that strain conventional infrastructures. These demands make genomics a prime candidate for scalable, cost-efficient, and resource-aware computing frameworks. This thesis explores this intersection through a case study based on OPERA-gSAM, a Spark-based implementation for high-throughput genomic data analysis using opportunistic computing.

This wide range of application heterogeneity depends on computing resources to process their jobs and efficiently deliver the service. Cluster computing is a prominent paradigm that integrates large-scale computing nodes to create powerful analytical environments, such as data centers, both locally and in the cloud. Prevailing Cluster Computing solutions often involve statically partitioning computing resources into application-specific clusters, leading to the creation of infrastructure silos within enterprises.

However, these standard usage patterns do not fully exploit most modern cluster computing power. Current resource managers also do not support dynamically sharing their capabilities, that is, computing power, network, and storage, to other resource managers. Further, they do not take advantage of on-premise capabilities within an enterprise's physical confines in an on-demand, elastic, or automated scaling approach. These concerns can result in low utilization and performance degradation, which reduces the overall performance of the system.

Estimates for 2021 place the contribution of the Information and Communication Technology (ICT) sector to global carbon emissions between 1.8% and 3.9%, placing it on par with the

aviation industry [4]. However, more recent assessments indicate that computing emissions have continued to increase, now reaching nearly 4% of total global greenhouse gas emissions, exceeding those attributed to the entire airline sector [5]. This trend underscores the increasing environmental impact of digital infrastructure and the growing urgency of sustainable computing practices.

Opportunistic computing leverages unused computing resources across a network to perform large-scale computations without requiring dedicated infrastructure. Over the years, many systems have been developed [6] [7] [8] to utilize these resources for high-performance or high-throughput computing tasks. This approach capitalizes on idle times in existing systems, such as personal computers and office workstations to achieve intensive computational workloads at a fraction of the cost of traditional methods. By effectively pooling and managing these sporadically available resources, opportunistic computing can handle intensive computational workloads at a fraction of the cost of conventional methods. HTCondor is a well-established system for managing and scheduling workloads across distributed computing environments [6]. Its ability to harness idle computing resources makes it particularly well-suited for implementing opportunistic computing strategies.

Aiming to address the above issues and provide a *unified data centre* for an enterprise, we propose BigOPERA, which stands for OPportunistically and Elastic Resource Allocation. BigOPERA is an enabling platform that can be used to take advantage of already installed High-throughput resources (`HTCondor`-based) for the benefits of High-performance (Hadoop-like) applications. Based on containerization technology, BigOPERA implements a container-based cluster on top of readily available computing components. Using BigOPERA, Docker containers are employed as worker nodes in an amalgamation approach that serves parallel computing (e.g., data-intensive applications). Our platform abstracts resource allocation into a fine-grained model, maximizing valuable computations for extensive data operations and enhancing the return on infrastructure investment.

We propose our scheduler, BigOPERA, for Apache Spark-based Big Data frameworks. BigOPERA capitalizes on the available and underutilized (idle or lightly used) capabilities within the physical confines of an enterprise. By scavenging these resources, BigOPERA keeps spawning Spark[9] DataNodes to benefit Big Data applications based on Docker containers as worker nodes. It is a standalone pluggable architecture that does not require modifying the existing Spark design. Additionally, we have developed a new scheduler version that offers direct support for hybrid architectures 3.2. By utilizing BigOPERA, an opportunistic

pool (by `HTCondor`) and a dedicated cluster can collaborate to achieve an enhanced task throughput with minimal cost of deployment. BigOPERA will launch disposable containers among the idle servers creating an opportunistic container-based environment, and ensure efficient provisioning for the Spark dedicated resources on demand.

1.1 Problem statement

With the advent of the BD era, practitioners have started building larger clusters using data lake/data hub architectures to support data-intensive applications. These clusters aim to accommodate a variety of dissimilar workloads for the daily data analytics and operations of the organization. Meanwhile, HTCondor is a leading and primarily used at High-Throughput Computing (HTC) facilities, such as particle accelerator research centres [10], with efficient resource-sharing across distributed nodes for the broader HTC applications.

Containerization technologies have revolutionized application deployment and management, addressing longstanding issues of resource inefficiency. With tools like Docker, which offer lightweight and isolated environments, applications have become significantly more scalable and more accessible to manage [11]. These advancements were pivotal in developing container orchestration systems such as Borg and Kubernetes [12].

Containers have changed how we build and deploy software. Bundling applications, configurations, libraries and dependencies in a consistent format allows software developers to create a new level of abstraction. These new containerized applications shifted the focus in data centres from hardware to the applications themselves, underscoring their far-reaching impact.

Moreover, the project aims to answer urgent questions concerning large-scale distributed clusters: utilization, flexibility, scalability, and performance. Table 1.1 illustrates the challenges and issues we are addressing with BigOPERA.

1.2 Motivation

The exponential growth of data-driven computing has intensified the need for infrastructures that are not only cost-effective and performance-optimized but also environmentally sustainable. As global attention shifts toward addressing climate change, organizations face increasing pressure to reduce their carbon footprint while meeting the computational demands of modern workloads.

Table 1.1: Defining BigOPERA's knowledge domain

Num.	Problem Description	Type
1	Low utilization of most modern commodity computers and clusters	Utilization
2	Resources static partitioning within a physical confine of an enterprise	Flexibility
3	Statically sizing the cluster on peak utilization and installing new servers to enhance the throughput when demand.	Flexibility
4	Increasing gap between computation and I/O capacity on high-end workstations for scientific experiments.	Performance
5	Dependency on multi-tenant (e.g., cloud computing) raises many issues, such as security, data movement, performance degeneration, etc.	Utilization
6	Wasting many computing cycles for testing middleware and applications that are not in production yet.	Utilization
7	The need to enhance the performance with minimal cost of deployment.	Performance
8	Running a new type of experiment on the same resources with minimal configuration and keeping it extensible to any prior design.	Flexibility
9	Reducing carbon footprint.	Optimization

Leading cloud providers have taken proactive steps toward environmental sustainability. Amazon Web Services (AWS) has committed to achieving net-zero carbon emissions by 2040, detailing concrete actions in its annual sustainability reports [13, 14]. Microsoft, carbon neutral since 2012, aims to become carbon negative by 2030, focusing on renewable energy, carbon removal, and sustainable procurement strategies [15, 16]. Similarly, Google reached carbon neutrality in 2007 and aspires to operate on 100% carbon-free energy by 2030 [17]. Its environmental reports emphasize progress in data center efficiency, renewable energy investment, and circular economy initiatives. These corporate sustainability efforts reduce ecological impact and align with shifting consumer expectations. Studies have shown that environmentally conscious consumers increasingly favor companies with strong sustainability practices [18].

Despite these advances, traditional data center operations continue to face challenges related to resource inefficiency. Many production environments suffer underutilization due to static resource allocation models and workload variability. Studies at companies like Twitter and Google report significant gaps between requested and actual resource usage, with CPU

utilization disparities reaching 53% and memory utilization up to 40% [19]. Overprovisioning to accommodate peak loads further exacerbates these inefficiencies, as cluster workloads fluctuate based on time, job type, and user activity. Weekday traffic often exceeds weekend demand, and multi-stage jobs introduce varied parallelism and resource requirements. These dynamic patterns result in idle computing capacity that could otherwise be repurposed.

Opportunistic computing offers a practical solution to inefficiencies in resource usage. Organizations can actively harness idle or underutilized resources—such as spare CPU cycles on desktops, unused server capacity, or discounted cloud instances—to reduce waste, increase throughput, and cut operational costs. At the same time, they lower their environmental footprint by improving energy efficiency and avoiding unnecessary infrastructure expansion. This sustainability-driven approach builds on the same motivation that inspired researchers in the late 1980s to utilize idle desktop machines in university settings to overcome hardware limitations 2.2

By integrating opportunistic computing with distributed frameworks like Apache Spark, organizations gain flexibility to scale resources dynamically in response to fluctuating demand. This synergy allows teams to optimize resource allocation, reduce idle time, and avoid overprovisioning while still meeting service-level agreements (SLAs).

Opportunistic computing not only enhances performance and resilience but also aligns directly with the goals of green computing. It encourages engineers and system architects to embed sustainability into computing infrastructure design, turning environmental responsibility into a core operational strategy.

1.3 Objectives

The main objective of this work is to enhance Apache Spark resource efficiency and performance by integrating opportunistic computing into a unified, hybrid framework. The specific objectives are:

- Design and implement a scheduling mechanism that enables Apache Spark to prioritize dedicated resources while opportunistically using idle or underutilized nodes. The scheduler must be compatible with Spark existing architecture and operate without requiring modifications to user applications.
- Develop a container-based hybrid framework (BigOPERA) that allows Spark and HTCondor to coexist on the same infrastructure. The framework must support dynamic

resource provisioning, fault tolerance, and seamless execution of Spark jobs across heterogeneous environments.

- Perform a performance evaluation using industry-standard big data benchmarks to assess the generalizability of the framework across diverse workloads.
- Evaluate the fault tolerance of the framework by simulating the loss of opportunistic nodes during execution and measuring the impact on job completion, recovery time, and data integrity.
- Evaluate the performance of the hybrid setup through a practical application in the field of genomics. This includes comparing execution times for sorting and UMI counting operations against native Spark and HPC baselines, using varying data sizes to assess scalability.

1.4 Thesis Structure

Chapter 2 presents the necessary background on Big Data, opportunistic computing, and existing related scheduling frameworks. Chapter 3 introduces the architecture, design principles, and scheduling mechanisms underpinning BigOPERA. In Chapter 4, performance evaluations are conducted using standard benchmarks and real-world workloads. Chapter 5 demonstrates the practicality of BigOPERA through the OPERA-gSAM use case in genomics. Finally, Chapter 6 summarizes the thesis findings, discusses current limitations, and outlines directions for future research.

CHAPTER 2

BACKGROUND

Resource management in Big Data environments is a critical area of research, particularly as data-intensive applications continue to grow in complexity and scale. This chapter provides the foundational context for understanding the motivations and design choices behind the BigOPERA framework. It begins by examining the evolution of Big Data processing systems, with a focus on Apache Spark and its architecture, followed by an overview of the limitations of conventional resource management strategies. The discussion then turns to opportunistic computing as a sustainable alternative for leveraging underutilized resources, presenting its historical roots and contemporary implementations. Additionally, the chapter reviews the energy impact of data centers and the emerging importance of green computing. Finally, a review of existing approaches highlights key efforts in hybrid resource scheduling, elasticity, and fault tolerance. These sections collectively ground our proposal and identify the research gaps that BigOPERA aims to address.

2.1 Big Data

The term “Big Data" was first coined in the mid-1990s by John Mashey, a retired Chief Scientist at Silicon Graphics, to describe the handling and analysis of large datasets [20]. The concept gained further clarity in 2001 when Doug Laney, an analyst at Meta Group, introduced the “3Vs” framework—Volume, Velocity, and Variety—which became the foundational definition for Big Data [21]:

- **Volume:** Represents the sheer scale of data produced and accumulated, often measured

in petabytes or exabytes.

- **Velocity:** Describes the real-time or near-real-time speed at which data is produced and must be processed.
- **Variety:** Denotes the diversity of data types, including structured, semi-structured, and unstructured data (e.g., text, images, videos, logs).

As data-intensive technologies have evolved, researchers and analysts have proposed additional characteristics to better describe the complexity and utility of Big Data [22]. These attributes aim to capture nuances in quality, relevance, and data dynamics:

- **Fine-grained and uniquely indexical:** Emphasizes high-resolution data with precise identifiers, often at the individual or object level [23].
- **Relationality:** Highlights how datasets often include shared fields that enable integration or linkage with other datasets [24].
- **Extensionality and Scalability:** Describes systems that can adapt and grow easily, with new fields or records added dynamically [25].
- **Exhaustivity:** Suggests that data now captures complete systems or populations ("n = all"), rather than relying on sampling [26].
- **Veracity:** Acknowledges the presence of noise, uncertainty, and error in datasets [27].
- **Value:** Recognizes the potential insights, strategic applications, or monetization derived from data analysis [27].
- **Variability:** Captures the shifting meaning of data depending on context, source, or time [28].

The continual expansion of the Big Data definitional framework has prompted critical debate. Industry analyst Seth Grimes has argued that many of the newer "Vs" do not represent the intrinsic properties of Big Data but some operational or contextual concerns [29]. Doug Laney, originator of the 3Vs, has publicly agreed with this assessment [30], warning against definitional inflation that obscures analytical clarity.

2.1.1 Big Data Systems

Big data refers to the analysis of large data sets that serve as a key basis for competition, driving innovation, growth in productivity, and consumer surplus in various sectors. It encompasses data that is enormous in volume, diverse in type, and has a complicated multidimensional structure, often generated from Internet applications and communications. This data requires specialized techniques for storage and analysis to provide accurate analytics reports and interesting data visualizations [31].

In management, Big Data allows executives to measure and manage more precisely, make better predictions, and target more effective interventions based on data rather than intuition [32]. It is considered far more potent than traditional analytics due to its ability to handle large volumes and the heterogeneity of data [33].

Big Data applications span various fields. For example, in healthcare, structured and unstructured data from sources such as databases, transaction records, emails, documents, devices, and sensors help to make data-driven decisions and advance medical facilities [34]. It is characterized by large size, high dimension, and complex structure in finance, significantly influencing decisions and predictions [35]. In agriculture, big data refers to the streaming data generated by Internet of Things (IoT) devices and social networks, offering new opportunities to monitor agricultural and food processes [36].

The era of Big Data has begun, with massive quantities of information produced and analyzed from various sources [37]. This data is utilized for decision-making and creating economic benefits by uncovering the potential value of data, providing strategies for social and economic development [38].

Big Data has revolutionized numerous fields by enabling the analysis of vast and complex datasets previously inaccessible or too cumbersome to manage. However, the benefits of Big Data are accompanied by several challenges that organizations and researchers must address. These challenges include, among others, data privacy and security concerns, data quality and integrity, scalability and infrastructure costs, technical complexity, and conducting a thorough cost-benefit analysis[2].

Reliable results depend heavily on the quality and consistency of data. Data used in Big Data systems often comes from unrelated sources, like sensor logs, social media feeds, and transaction records. Because of this, it is common to encounter problems like missing values, duplicates, or conflicting entries. Cleaning and validating this kind of data takes time and often requires domain knowledge to get it right. Making sure the data is trustworthy also

means knowing where it came from and being able to spot patterns that might suggest bias or error. As the amount of data keeps growing, many organizations find it hard to keep up, especially when their systems were not built to handle data at this scale.

Scalability involves hardware capabilities, such as storage and processing power, and software solutions, including distributed computing frameworks like Hadoop and Spark[39]. However, scaling infrastructure incurs substantial costs, from initial capital expenditure on hardware to ongoing operational costs for maintenance, energy consumption, and cloud services. Scaling up Big Data systems gets expensive fast. More data means more machines, more storage, and more power. Costs that not every organization can absorb.

The technical complexity of Big Data systems presents another challenge. Implementing and managing Big Data platforms requires specialized skills in data engineering, machine learning, and database management. Integrating diverse data sources, real-time data processing, and developing scalable algorithms demands technical proficiency.

BigOPERA explicitly aims to alleviate these two key challenges. This work tackles the problem by reusing hardware that organizations already have, rather than requiring new infrastructure. It helps enterprises scale up when needed without adding too much complexity. The system improves performance and ensures the infrastructure remains manageable and user-friendly, which is important for teams that need to keep things running without a lot of overhead.

One significant technological shift in next-generation parallel and distributed computing is in BD platforms. BD addresses the increasing gap between the rapid growth of data size and computational capacities. Motivated by data lakes that currently exceed zettabytes of data and a large community behind it, BD is rapidly evolving new processing frameworks. These frameworks accommodate a variety of scenarios for different markets and data types. Apache Spark has become the de facto platform for widespread BD applications as a unified engine for diverse workloads. Using Spark, an implementer may store their data in distributed file systems (e.g., HDFS, HBase, etc.) and then employ an execution framework (micro-batch, SparkSQL, stream, etc.) to process this large-scale data. The brain behind this phenomenon is an efficient abstraction of in-memory cluster computing called Resilient Distributed Dataset [40].

Big Data has transformed how we collect, process, and use information, yet it brings numerous challenges and disadvantages that demand careful attention. One of the primary issues is data privacy and security. The more information that a company stores, the harder it becomes to secure against bad actors. Data leaks, identity theft, and privacy violation risks

necessitate robust security measures and strict regulatory compliance.

Ensuring data quality and accuracy is another significant hurdle. Big Data often contains noise, inconsistencies, and errors, leading to misleading analyses and decisions. Addressing the trustworthiness of data involves thorough cleansing, validation, and quality control processes.

The volume of Big Data requires substantial storage, processing, and infrastructure investments. Scaling systems to handle growing datasets can be costly, and maintaining these systems involves continuous financial commitments. Working with Big Data often involves a level of technical complexity that many teams aren't prepared for. Older data tools usually can't handle the scale or speed required, so teams need to rely on newer technologies built for distributed systems. This shift demands people who understand how to manage large-scale storage, parallel processing, and machine learning pipelines, skills that take time to learn and are often in short supply. As a result, getting these systems up and running can be challenging.

Integrating Big Data from diverse sources, such as social media, sensors, and legacy systems, is complex. Data silos and compatibility issues can obstruct efforts to gain a comprehensive view of the data, limiting the insights that can be derived.

2.1.2 Map Reduce

In 2004, while working for Google, Jeffrey Dean and Sanjay Ghemawat developed a programming model and distributed execution framework designed to process large amounts of data across thousands of machines efficiently. This model took inspiration from the *map* and *reduce* directives present in several functional languages and was designed to be highly scalable and run on large clusters of commodity hardware, without the need for specialized hardware [41].

MapReduce is designed to simplify large-scale data processing across distributed systems. It abstracts the complexities of parallelization, fault tolerance, data distribution, and load balancing, allowing programmers to focus solely on the computation logic. The model processes data in two main phases: Map and Reduce, and intermediate steps that optimize performance, as shown in Figure 2.1.

- Map phase: The computation begins with the Map function, which processes input key-value pairs independently and emits intermediate key-value pairs.
- Shuffle and sort phase: After the Map phase, the system groups all intermediate values by their keys and sorts them. This ensures that all values associated with the same key

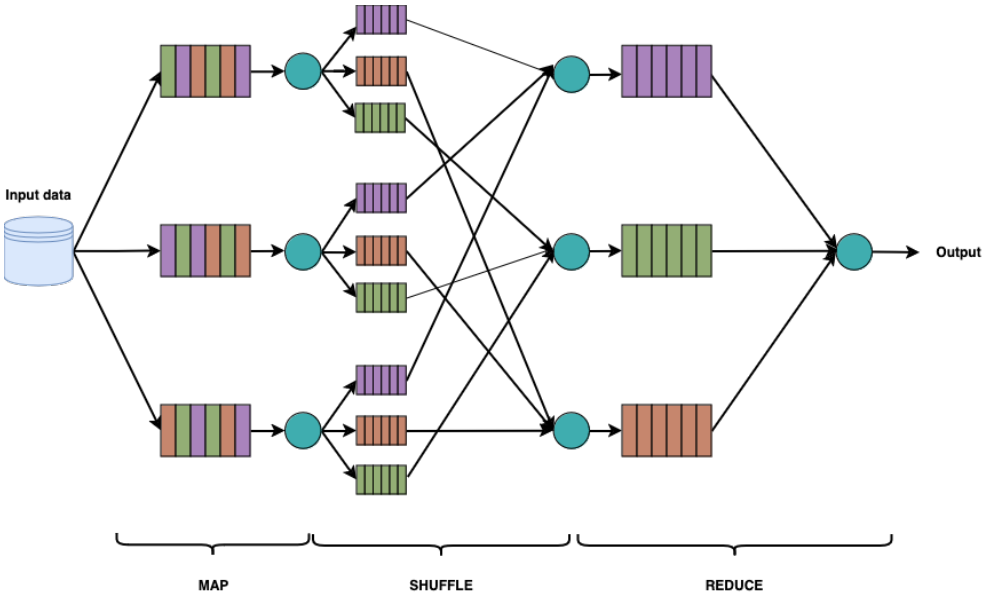


Figure 2.1: Map reduce algorithm

are passed to the same Reduce task.

- Reduce phase: The Reduce function processes each grouped key and its list of values, merging them into a final result.

A central advantage of this approach is the abstraction of parallelization and distribution complexities, allowing users to focus on program logic while the framework manages infrastructure. This design enhances both usability and scalability in distributed environments.

2.1.3 Hadoop

Google MapReduce, in addition to the Google File System paper [42], served as inspiration for Dog Cutting and Mike Cafarella (then working at Yahoo! and University of Michigan, respectively) to develop Hadoop under the umbrella of the Apache Foundation. The official project page defines Hadoop as: *a framework that allows for the distributed processing of extensive data sets across clusters of computers using simple programming models. It is designed to scale from single servers to thousands of machines, each offering local computation and storage* [43].

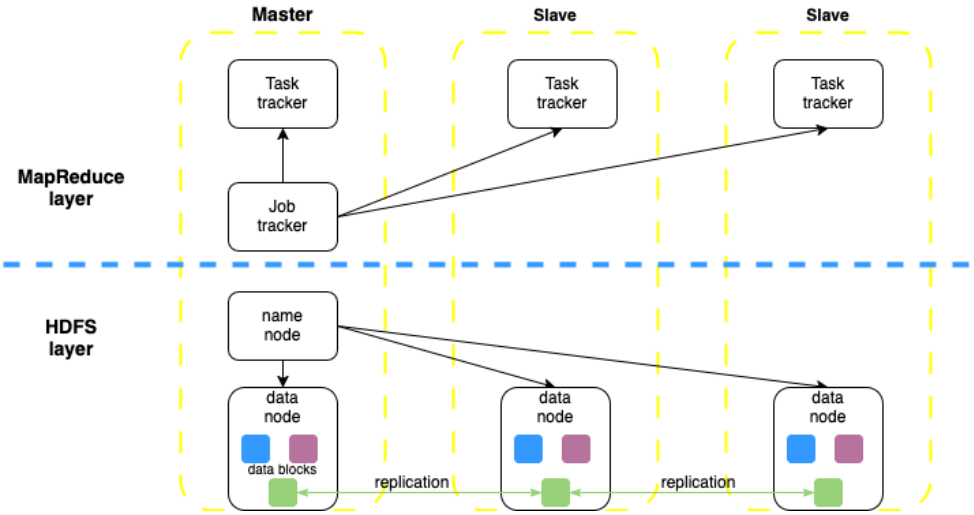


Figure 2.2: Hadoop architecture

At its core, Hadoop consists of two primary components: the Hadoop Distributed File System (HDFS) for reliable data storage and Hadoop MapReduce for parallel data processing. HDFS is the storage backbone of Hadoop, designed to handle massive files by splitting them into smaller, manageable blocks distributed across multiple machines [44]. A typical Hadoop cluster, depicted in Figure 2.2, follows a master-slave architecture for HDFS, where a single NameNode acts as the master, maintaining the filesystem metadata, such as directory structure, file permissions, and the mapping of file blocks to DataNodes. Data is stored and managed by DataNodes, which serve as worker nodes. Each file in HDFS is divided into fixed-size blocks (traditionally 128 MB or 256 MB), and each block is replicated across multiple DataNodes (usually three) to ensure fault tolerance. If a DataNode fails, the system automatically retrieves data from another replica, maintaining uninterrupted access. Additionally, DataNodes periodically send heartbeat signals to the NameNode to confirm their availability, allowing the master to detect and recover from failures.

Hadoop utilizes the MapReduce programming model, introduced in Section 2.1.2, to process data in parallel across distributed clusters. In earlier versions of Hadoop (1.x), job scheduling and task execution were managed by a JobTracker on the master node, which assigned tasks to TaskTrackers running on worker nodes. However, this architecture had limitations in scalability and flexibility, leading to the introduction of YARN (Yet Another

Resource Negotiator) in Hadoop 2.0 [45].

Beyond its core functionalities, Hadoop has expanded into a rich ecosystem, incorporating tools for data warehousing (e.g., Apache Hive), distributed coordination (e.g., Apache Zookeeper), and scripting (e.g., Apache Pig). Apache Hadoop remains actively developed and widely adopted in industry. Companies like LinkedIn rely on it for large-scale data processing [46]. At the same time, Uber operates one of the world's largest Hadoop deployments, managing over 1 exabyte of data across tens of thousands of servers in its two geographically distributed regions [47].

2.1.4 BD with Apache Spark

The growing demand for large-scale data processing revealed significant limitations in existing frameworks like MapReduce, particularly for workloads requiring repeated access to datasets. Traditional systems relied on an acyclic data flow model that forced data to be reloaded from disk for each computation cycle, creating substantial overhead for iterative algorithms and interactive queries. Zaharia et al. introduced Apache Spark in 2010 to overcome these inefficiencies by enabling efficient handling of workloads with reusable datasets through in-memory computation and fault-tolerant data structures [48]. Spark key innovation was its ability to handle workloads with working sets efficiently, datasets reused across multiple parallel operations, through a combination of in-memory computation and fault-tolerant data structures.

At the heart of Spark design are resilient distributed datasets resilient distributed datasets (RDDs), representing a fundamental shift in managing distributed computations. RDDs are immutable, partitioned collections of objects that can be reconstructed if lost, thanks to their lineage—a record of the transformations used to build them. This approach provides fault tolerance without the overhead of traditional checkpointing, as only lost partitions need recomputing, and this can be done in parallel across the cluster. Users can explicitly cache RDDs in memory, enabling rapid reuse across multiple operations while maintaining the system's ability to fall back to recomputation when memory is constrained, much like virtual memory systems.

What set Spark apart was its ability to support complex workflows that previous systems struggled with. Iterative machine learning algorithms, which repeatedly apply functions to optimize parameters, could run orders of magnitude faster by keeping working datasets in memory between iterations. Similarly, interactive data analysis became feasible, with users

able to query multi-gigabyte datasets with sub-second response times after the initial load. These capabilities were complemented by Spark tight integration with the Scala programming language, which allowed developers to write concise, functional-style code while benefiting from the framework distributed processing power.

The implementation cleverly leveraged existing infrastructure, building the Mesos cluster manager atop while introducing novel abstractions. Spark's architecture treated RDDs through a simple interface of partitions, iterators, and preferred locations, enabling efficient task scheduling and data locality optimization. The system also introduced specialized shared variables—broadcast variables for efficient distribution of read-only data and accumulators for reliable distributed aggregation—that expanded the range of expressible algorithms while maintaining fault tolerance.

Performance results demonstrated Spark transformative potential. In comparative tests, Spark ran iterative machine learning workloads up to ten times faster than Hadoop by eliminating disk I/O between iterations. Its ability to support interactive exploration of a 39 GB Wikipedia dataset with sub-second query latencies after the initial load was even more impressive. These capabilities, combined with Spark's elegant programming model and fault tolerance characteristics, marked a significant advance in cluster computing. The framework success in bridging the gap between batch processing and interactive analysis would later influence an entire ecosystem of tools for large-scale data processing.

Spark design represented both a technical and conceptual breakthrough. While drawing inspiration from MapReduce and systems like Dryad, it introduced abstractions that generalized beyond their limitations. The notion of RDDs with lineage-based recovery provided a sweet spot between expressiveness and reliability, enabling new classes of applications while maintaining the scalability and fault tolerance that had made earlier systems successful. This work laid the foundation for what would become one of the most widely used frameworks in big data analytics, shaping the evolution of distributed data processing for years to come.

2.2 Opportunistic computing

Opportunistic computing is a paradigm that harnesses idle computational resources across a distributed network, thereby enabling large-scale task execution without the need for dedicated hardware infrastructures. This model is particularly advantageous in academic and research environments, where significant computational resources often remain underutilized outside

peak usage hours. By transparently aggregating these surplus resources, opportunistic computing fosters an elastic, high-throughput environment conducive to data- and compute-intensive workloads.

The concept of opportunistic computing originated in the late 1980s [49, 50, 51], when researchers sought to address resource constraints by utilizing idle desktop machines in university environments. One of the first systems to formalize this model was HTCondor (originally "Condor"), developed at the University of Wisconsin-Madison by Miron Livny and colleagues[52]. HTCondor was explicitly designed to exploit underused computing cycles on workstation clusters by providing a framework that could execute jobs on machines not in active use, reschedule jobs upon resource preemption, and migrate them as needed. This early innovation laid the foundation for modern high-throughput computing (HTC) by demonstrating that substantial computational power could be obtained without dedicated infrastructure.

HTCondor enables users to submit computational jobs to a pool of heterogeneous machines, which execute these jobs when otherwise idle, without disrupting primary users. The system supports job queuing, priority-based scheduling, checkpointing, and remote execution, making it well-suited for HTC environments. HTCondor can integrate with grid infrastructures and other batch systems, extending its flexibility and scalability [53].

In the context of data-intensive scientific applications, HTCondor has been instrumental in enabling large-scale experiments and simulations. It has been adopted in projects such as the Open Science Grid (OSG), where it facilitates the execution of millions of jobs per day across geographically dispersed sites [54]. Its matchmaking algorithm and ClassAd mechanism allow precise resource-job pairing, supporting advanced job placement policies based on availability, load, and other user-defined constraints [55].

The opportunistic use of idle desktops, lab machines, and laptops via HTCondor illustrates a sustainable approach to distributed computing. Institutions can significantly amplify their computational capacity without incurring additional hardware costs or energy consumption by turning otherwise passive infrastructure into a productive computing substrate.

2.3 Server Power Consumption and Energy Usage Patterns

Analyzing the energy consumption of server computers requires considering multiple components, including cooling, storage, networking, memory, and multi-socket CPUs. A survey from 2016 indicates that cooling accounts for approximately 50% of energy consumption,

with storage and networking each consuming around 10% [56]. Another study conducted in 2020 reports that CPU consumption constitutes about 32% of the total power use in data centers [56].

The instantaneous power consumption of a server can be described through an additive model, combining contributions from the CPU, hardware accelerators such as GPUs, RAM, storage, network interface cards (NICs), cooling mechanisms, and auxiliary electronic components [57]. This relationship can be expressed mathematically as follows:

$$P_{system,t} = P_{CPU,t} + P_{Accel,t} + P_{RAM,t} + P_{Disk,t} + P_{NIC,t} + P_{Cool,t} + P_{Aux,t} \tag{2.1}$$

Each power component varies over time depending on the utilization level and the operational state (on or off). For instance, CPU-intensive workloads significantly raise $P_{CPU,t}$, whereas GPU-related tasks predominantly affect $P_{Accel,t}$. The increase in power from these workloads starts from a baseline consumption level, reflecting the static power used during idle periods. Consequently, the power component for any device decomposes into static and dynamic parts:

$$P_{device,t} = P_{dynamic,t} + P_{static,t} \tag{2.2}$$

This formulation aligns with a basic activity model [57]. Moreover, introducing power domains, which can be turned off to save energy, enhances this model, yielding:

$$P_{device,t} = P_{dynamic,t} + u \cdot P_{static,t} + (1 - u) \cdot P_{suspend,t} \tag{2.3}$$

Here, u is a binary state variable indicating whether the device operates (1) or remains suspended (0).

Experimental evidence confirms that baseline power consumption—the amount of energy consumed simply by having the server turned on—represents a substantial portion of total server energy usage [57]. The CPU component exerts the most significant influence on total consumption among all server components, but this influence does not scale linearly with CPU load. Disk I/O operations represent the second most significant contributor to energy use; their efficiency largely depends on the block size used in these operations [57]. Additionally, while network activity contributes less significantly, it still affects power consumption, scaling nearly linearly with network transmission rates. Other server components, including RAM and auxiliary electronics, typically remain accounted for within baseline consumption.

CPU energy consumption primarily depends on active core counts, CPU frequency, and processing load (measured in Active Cycles per Second, ACPS). Measurements show that power consumption increases linearly with load when only one core operates at a constant frequency. However, with multiple cores and variable loads, the consumption follows a concave, polynomial pattern [57]. Minimizing power consumption at a given load involves selecting the optimal combination of core count and CPU frequency—specifically, activating more cores at lower frequencies whenever feasible. The minimal power achievable for each load follows a piecewise concave relationship, rather than a simple linear one.

Energy efficiency, defined as active processing cycles per unit of energy, similarly benefits from tuning operational parameters. Maximizing this efficiency requires selecting optimal CPU core counts and frequencies tailored to the workload. Observations indicate that maximal energy efficiency occurs neither at the highest nor the lowest CPU frequencies available but at intermediate values with full utilization of available cores [57].

These insights suggest that carefully adjusting server operational parameters substantially reduces energy usage and improves overall data center efficiency.

2.4 Green computing

As digital infrastructures become increasingly integral to global economies, the environmental implications of computing, particularly energy consumption, have come under scrutiny. Green computing, also called sustainable computing, addresses the design, use, and disposal of computing systems in an environmentally responsible manner. As defined by Aruna and Padmavathi, it involves "the study and practice of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems—such as monitors, printers, storage devices, and networking and communications systems—efficiently and effectively with minimal or no impact on the environment" [58]. Central to this initiative is energy efficiency, which has emerged as a critical challenge and opportunity for stakeholders ranging from cloud providers to government policymakers. The proliferation of Internet of Things (IoT) devices and applications further amplifies the urgency of green computing, as billions of energy-constrained, interconnected devices increasingly contribute to global power consumption and electronic waste. The rapid expansion of computing infrastructure has incurred a considerable environmental cost. Emissions from the computing sector already account for nearly 4% of global greenhouse gas emissions—surpassing those of the entire airline industry. Without

significant shifts toward sustainable practices, projections suggest that 2040 computing alone could consume over half of the world allowable carbon budget, rendering current trajectories environmentally untenable [5].

Data centers are a focal point of energy consumption within the computing ecosystem. According to the 2024 report by Lawrence Berkeley National Laboratory, U.S. data centers consumed approximately 196 terawatt-hours (TWh) in 2022, accounting for about 4% of the country's total electricity use. This figure is expected to rise substantially with the rapid expansion of AI workloads and hyperscale cloud infrastructure [59]. Despite these increases, improvements in energy efficiency, such as better cooling systems, server utilization, and workload distribution, have tempered what might otherwise be exponential growth.

Supporting this trend, the International Energy Agency's 2025 *Global Energy Review* highlights that global energy consumption rose more rapidly than average in 2024, with electricity demand increasing nearly twice as fast as total energy demand. The rise in energy demand can be traced to several factors, including increased cooling requirements, broader industrial activity, the shift toward electric transport, and the rapid growth of data centers and AI-related computing [60]. These trends point to a clear connection between digital infrastructure and environmental impact, highlighting the need to take carbon emissions from computing more seriously.

The 2014 paper by Aruna and Padmavathi also outlines early recommendations for energy-efficient computing, highlighting virtualization, efficient power supplies, and workload consolidation as key strategies. These principles underpin modern data center operations and remain relevant in today's rapidly evolving digital infrastructure [58]. As the Green Computing book jokingly notes, the goal is to encourage data centers and supercomputing facilities to simulate climate change rather than "create it" [61]. Some of the largest cloud companies have started shifting toward more sustainable operations by investing in renewable energy and setting targets for carbon neutrality. Section 1.2 looks more closely at what these providers are doing in practice.

On the policy front, government-backed computing initiatives also play a significant role. The Trump administration's "Stargate 2025" project [62], which aims to boost American dominance in quantum computing and AI, carries substantial energy implications. Although primarily framed as a technological and geopolitical maneuver, the project includes the development of large-scale data centers with significant electricity requirements [63]. While public statements accompanying the initiative allude to streamlining energy infrastructure and

supporting AI development, environmental experts have warned that such efforts must be aligned with national and global climate targets to be sustainable [64]. This example shows how efforts to advance computing can conflict with environmental goals. It also points to the importance of treating energy efficiency as a core part of system design, rather than something to add later.

Opportunistic computing offers an additional path toward sustainable digital infrastructure by leveraging idle or underutilized resources, such as desktop machines or edge devices, to perform distributed workloads without deploying new dedicated hardware. This approach reduces the need for continuous operation of energy-intensive data center servers, thereby minimizing energy consumption and carbon emissions. By repurposing existing infrastructure, opportunistic computing aligns well with green computing principles and complements broader efforts to improve resource utilization and reduce environmental impact.

Given the scale of energy usage in computing, particularly in data centers, improving energy efficiency is not only an environmental imperative but also an economic and strategic one. Without significant advances in energy-aware design and sustainable infrastructure, the growth of cloud computing, AI, and quantum technologies may become incompatible with climate resilience goals. Therefore, Green computing must remain a foundational priority for industry and policy in the digital age.

2.5 BD Challenges

BD has been transformative across many fields, letting us analyze massive, complex datasets that were once too much to handle. However, this wealth of information comes with its problems that organizations and researchers need to tackle. These challenges include, among others, data privacy and security concerns [65], data quality and integrity, scalability and infrastructure costs, technical complexity, and the need for specialized expertise to manage and analyze large datasets effectively [2]. Setting up these systems usually requires a lot of configuration and adjustment to get them running smoothly, especially at scale.

As data volumes grow, many organizations struggle to expand their infrastructure fast enough to keep up with the demands of processing and managing that information. Ensuring that the infrastructure can handle increasing amounts of data involves enhancing hardware capabilities, such as storage and processing power, and utilizing distributed computing frameworks like Apache Hadoop [66] and Apache Spark [39]. However, this scalability comes

with substantial infrastructure costs. These costs range from the initial capital investments in hardware to ongoing expenses for system maintenance, energy consumption, and cloud services. Balancing the need for scalability with budget constraints is a critical consideration for organizations leveraging BD.

BigOPERA explicitly aims to alleviate the last two key challenges by providing a seamless and straightforward solution that integrates effortlessly with existing Spark clusters. It does not require Spark architecture, applications, or cluster configuration modifications. Unlike other solutions that demand additional libraries, dependencies, or infrastructure changes [67, 68], BigOPERA functions as an add-on that works with the existing Spark ecosystem. It enables enterprises to efficiently scale their extensive data operations without costly infrastructure upgrades.

2.6 Resource management

Cluster resource management has two primary responsibilities: allocating resources and scheduling tasks. Resource allocation is the process of distributing computing resources among users or applications based on a set of rules designed to maintain fairness or meet priority needs. To make the most of available resources, these rules often incorporate fairness principles or assign priority levels based on workload demands. In contrast, task scheduling determines where each unit of work, be it a task, process, container, or virtual machine, should run within the cluster. This decision takes into account the current state of the system to maximize performance and minimize delays.

Traditional approaches in data centers often partition computational resources into static, application-specific clusters. Zaharia et al. [69] highlight that while this method may provide predictability, it leads to inefficient resource utilization due to its rigid, preallocated structure. The inherently dynamic and fluctuating nature of BD workloads often conflicts with such static partitioning, resulting in computational inefficiencies and resource underutilization.

The emergence of the MapReduce programming model [70] and its open-source implementation, Hadoop, has spurred significant research interest [71]. In cluster computing, scheduling tasks efficiently has emerged as a central challenge, prompting extensive research and a variety of proposed solutions. Most of these efforts fall into one of two main categories [72].

- **Static (offline) scheduling**: makes all task-to-processor assignments before execution starts. This method depends on having detailed knowledge about how long each task

Table 2.1: Comparative analysis of different works in BD Resource Management.

Paper	Scal	Elast	R-A	Perf	C-E	Usab	Interop	Adapt
Mesos [76]	5	3	5	4	4	3	5	5
K8s [11]	3	5	5	5	4	4	5	5
Moon [67]	2	3	3	3	5	3	3	3
Sparrow [77]	4	2	4	4	4	3	3	3
BigOPERA	4	4	5	5	5	4	4	5

will take and what resources are available, allowing for a fixed plan to be drawn up in advance. While straightforward, it lacks flexibility in responding to runtime changes.

- **Dynamic (online) scheduling**: assigns jobs to processors during execution, requiring minimal prior knowledge of job resource requirements. This makes it well-suited to environments where workloads are unpredictable or constantly changing.

Dynamic scheduling is crucial in optimizing performance and efficiency in BD processing environments [73]. In contrast to static scheduling, which depends on fixed resource assignments made ahead of time, dynamic scheduling continuously adapts to shifting workload demands. Advanced algorithms, including ML-based techniques, have been employed to predict workload patterns and adapt resource allocations accordingly [74, 75].

Additionally, dynamic scheduling enhances resource utilization by reallocating idle or underutilized resources, maximizing throughput and minimizing latency. Due to its inherent flexibility and responsiveness, dynamic scheduling is particularly well-suited for managing the complexity and unpredictability of BD applications, leading to more efficient and resilient data processing systems. A comparative analysis of various BD resource management approaches is presented in Table 2.1. The table includes scores ranging from 1 to 5, reflecting how effectively each referenced work addresses key issues such as Scalability, Elasticity, Resource Allocation Efficiency, Performance, Cost-Effectiveness, Usability, Interoperability, and Adaptability. These scores were calculated based on the degree to which each system or framework proposes concrete strategies for managing critical aspects of Big Data environments. In the following, we describe these key issues and justify the scores assigned to each system:

- **Scal, Scalability:** Describe or score how well the paper addresses scalability issues, including handling larger datasets, parallel processing capabilities, etc. Mesos supports large clusters efficiently; K8s suits moderate scaling with containers; Moon focuses

on smaller datasets; Sparrow offers efficient scheduling for distributed environments; BigOPERA integrates nodes for scalability.

- **Elast, Elasticity:** Include insights or scores related to the system's ability to dynamically scale resources up/down based on workload. Mesos offers basic elasticity; K8s excels with auto-scaling; Moon requires manual scaling; Sparrow offers limited elasticity; BigOPERA dynamically adjusts resources.

- **R-A, Resource Allocation:** Provide a description or score illustrating the effectiveness, strategy, and efficiency of resource allocation discussed in the paper. Mesos employs sophisticated strategies; K8s efficient resource allocation; Moon uses static quotas; Sparrow emphasizes batch processing; BigOPERA merges opportunistic resources effectively.

- **Perf, Performance:** Comment or rate the performance attributes like throughput, latency, and job completion time as illustrated in the paper. Mesos provides high performance with varied latency; K8s provides high performance with advanced scheduling; Moon performance may vary due to opportunistic resources; Sparrow provides good performance in batch processing; BigOPERA increases existing cluster performance.

- **C-E, Cost-Effectiveness:** Elaborate or rate the cost management and resource utilization economics discussed or exhibited in the paper. Mesos shares resource pools economically; K8s is cost-effective for containers; Moon minimizes costs by design; Sparrow reduces idle times; BigOPERA maximizes idle resource use.

- **Usab, Usability:** Describe or score the user-friendliness, deployment ease, and management simplicity portrayed in the paper. Mesos can be complex; K8s includes an expansive tool ecosystem; Moon usability may be limited due to its model; Sparrow targets technical users; BigOPERA is user-friendly within existing workflows.

- **Interop, Interoperability:** Share insights or scores about how well the proposed system can be integrated with existing tools and platforms. Mesos supports multiple frameworks; K8s integrates across systems; Moon is standalone; it is specific to distributed systems; BigOPERA adapts to existing platforms.

- **Adapt, Adaptability:** Provide descriptions or scores related to how the proposed system manages changing environments and diverse workloads. Mesos adapts to cluster

changes; K8s handles diverse workloads; Moon adaptability may be limited due to its opportunistic nature; Sparrow offers adaptability for specific use cases; BigOPERA efficiently manages dynamic environments.

The elasticity of resource allocation has also been extensively explored in the context of cloud and distributed computing environments. Verma et al. [78] emphasize the advantages of elastic resource allocation, highlighting its ability to improve computational throughput by dynamically adjusting resource availability in response to workload fluctuations. However, a persistent challenge in elastic resource allocation is ensuring that resources are optimally distributed in alignment with workload demands. Addressing these critical challenges, this work introduces a two-tiered, fine-grained scheduling mechanism (BigOPERA) designed to enhance resource efficiency and adaptability in BD environments.

2.7 Related Work

The paradigm of dynamically creating disposable clusters, as espoused by BigOPERA, finds echoes in the work of Hindman et al. [76], wherein Mesos is introduced as a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, ensuring optimal resource utilization by sharing resources in a fine-grained manner among different applications. Similarly, MOON [67] introduces a novel approach to leveraging opportunistic computing environments for MapReduce workloads. This approach extends the concept of dynamic cluster creation by harnessing transient and opportunistic resources and optimizing resource utilization for distributed data processing tasks. The work described in [67] highlights the importance of adapting to changing resource availability and aligns with the broader trends in dynamic cluster management for efficient, cost-effective, and scalable distributed computing. Another notable contribution to task scheduling in distributed data processing systems is Sparrow [77]. Unlike centralized schedulers that struggle with latency and throughput under highly parallel, short-lived workloads, Sparrow proposes a decentralized, stateless scheduling architecture designed for low-latency task assignment at scale. It uses techniques such as batch sampling and late binding to improve task placement while maintaining scalability and fault tolerance. By distributing scheduling decisions across multiple independent schedulers, Sparrow avoids common bottlenecks and provides near-optimal response times, making it a compelling solution for fine-grained scheduling in large clusters. While its focus lies in

latency-sensitive workloads rather than elasticity or opportunistic computing, its design goals align with BigOPERA emphasis on scalability and dynamic adaptability.

Kubernetes [12] has become a de facto standard for container orchestration in modern cloud-native environments, offering robust support for deploying, scaling, and managing containerized applications. In the context of Big Data processing, several works have explored integrating Apache Spark with Kubernetes to leverage its elasticity, fault tolerance, and declarative configuration model. Kubernetes enables dynamic resource provisioning, making it suitable for heterogeneous and transient workloads. However, its scheduling model prioritizes container lifecycle management and resource isolation, rather than fine-grained task-level optimizations needed in distributed data analytics. As such, while Kubernetes simplifies the operational management of Spark clusters, it does not natively support opportunistic resource scheduling or dynamic executor preemption. BigOPERA complements this gap by introducing an opportunistic scheduling mechanism that operates independently of Kubernetes' internal scheduler, making it adaptable to both containerized and non-containerized execution environments.

The increasing adoption of Apache Spark has brought resource utilization to the forefront of research. Both industry and academia have made significant efforts to enhance Spark's performance and efficiency, such as the TR-Spark project [79]. Other studies, particularly those focusing on cloud environments, have worked on improving Spark's resilience by optimizing the use of *spot* instances [80] and focusing on cost savings through better resource utilization [81].

OpERA [68] represents a significant effort to enhance resource utilization within BD frameworks by modifying the Fair Scheduler of Hadoop Yarn [82]. The study proposes a strategy to increase resource utilization and reduce job execution times by reallocating available resources to pending tasks based on runtime utilization knowledge. The central concept is to improve resource sharing; the scheduler can then reallocate idle or underutilized resources from one task to another waiting task. This creates a more granular scheduling approach to optimize resource use within the dedicated resource pool. In contrast, BigOPERA extends this concept by incorporating non-dedicated workers, thus expanding resource utilization beyond the confines of the cluster's dedicated resources.

Our work distinguishes itself by proposing a hybrid model approach for Apache Spark, which, to our knowledge, remains unexplored in current literature. Moreover, our solution can be universally applied across cloud and local cluster systems.

CHAPTER 3

BIGOPERA FRAMEWORK

This chapter presents the methodology followed in the design and implementation of BigOPERA. It details the architectural components of the system, including the motivation behind key design decisions, the role of containerization for runtime consistency, and the integration of non-dedicated nodes. The chapter also describes the two-tiered scheduler that prioritizes dedicated resources while opportunistically utilizing idle ones, along with the internal algorithms responsible for managing task assignment, node state transitions, and error handling.

3.1 Architecture and design

The BigOPERA architecture integrates a hybrid computing environment combining HTCondor for opportunistic computing with Apache Spark for distributed data processing and Kubernetes for managing dedicated resources. This dual-faceted approach allows for dynamic workload execution by seamlessly integrating both dedicated and non-dedicated computing resources, optimizing both cost and performance.

The architecture visually depicted in Figure 3.1 is divided into two main sections: the Opportunistic Pool and the Dedicated Pool, highlighted by green and purple dashed lines, respectively.

The Opportunistic Pool, managed by HTCondor, consists of non-dedicated nodes that execute Spark tasks opportunistically whenever spare computational resources become available. These nodes must support Docker containers and maintain connectivity to the dedicated network directly or via VPN. As nodes become available, they automatically register with the

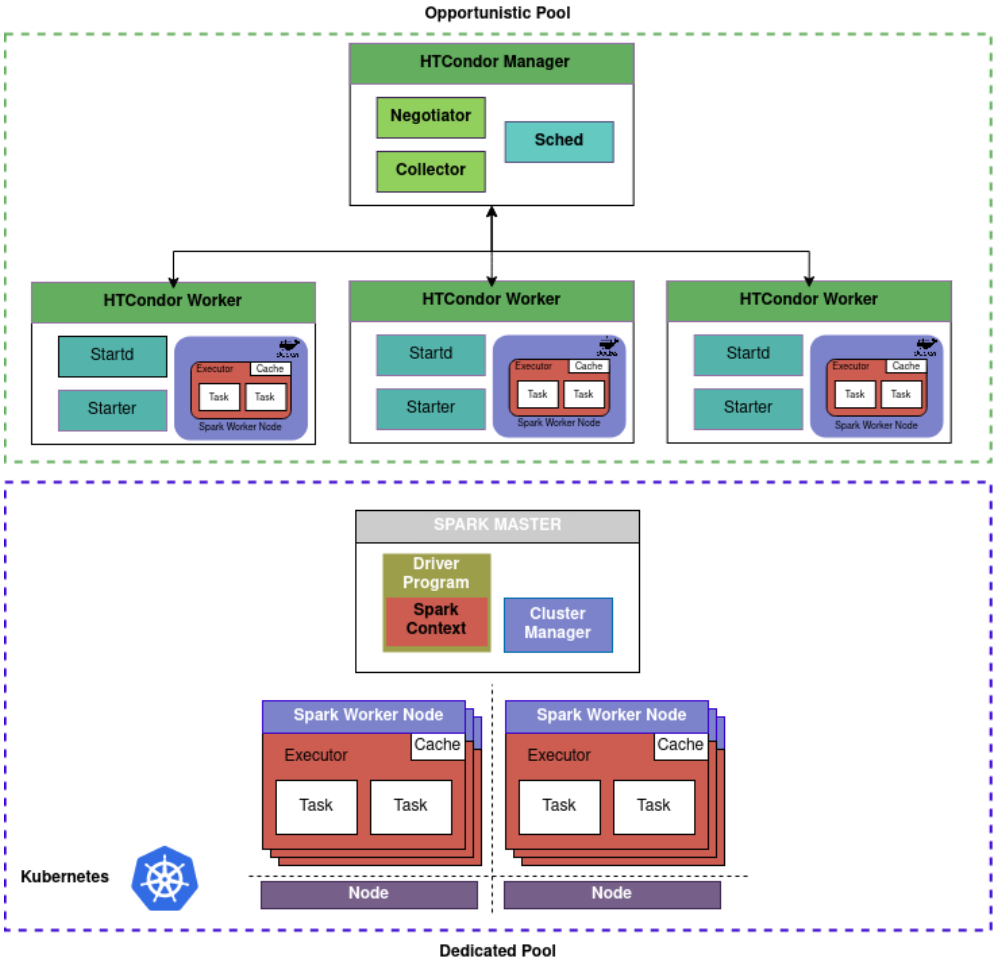


Figure 3.1: BigOPERA architecture.

HTCondor Manager, signalling their readiness to run Spark jobs. This dynamic registration enables the architecture to scale flexibly and efficiently based on resource availability.

The HTCondor Manager, including components like the Negotiator, Collector and Scheduler (Sched), oversees job scheduling and resource allocation. Worker nodes in this pool are equipped with Startd and Starter daemons (shown in green color), and they run Spark Worker Nodes capable of executing tasks dynamically. These Spark workers (purple color) utilize Docker and control groups (cgroups) to run multiple tasks concurrently, efficiently utilizing

available resources.

HTCondor daemons periodically report node status and resources to the central manager, providing essential data such as CPU, memory, disk usage, and network availability. This information is crucial for making informed scheduling decisions and ensuring efficient task execution.

Conversely, the Dedicated Pool operates under Kubernetes, ensuring stable computational capacity for Spark workloads. Here, the Spark Master orchestrates job execution through its Driver Program, Spark Context (red color), and Cluster Manager, distributing tasks across multiple Spark Worker Nodes (purple). Each worker node hosts Executors to run Spark tasks and Caches to store frequently accessed data, thereby enhancing processing efficiency.

Spark Worker nodes within this architecture periodically report their available resources (such as CPU and memory) to the Spark Master, which is responsible for assigning tasks and launching executors accordingly. This direct interaction between the Spark Master and Worker nodes enables efficient resource utilization and dynamic task execution across the cluster.

The combination of these two pools allows the architecture to balance cost and performance effectively. HTCondor, with its capability to manage opportunistic resources and Docker containerization, complements Spark Standalone lightweight resource management and task execution model. Together, they create a resilient, scalable, and efficient environment for running large-scale data analytics workflows with Apache Spark, dynamically leveraging dedicated and available non-dedicated resources. This comprehensive architecture, by integrating both opportunistic and dedicated computing resources, provides a flexible and scalable solution for handling extensive data processing tasks efficiently.

3.1.1 Design Rationale and Key Architectural Decisions

The design of BigOPERA was shaped by the goal of enabling scalable, fault-tolerant data processing while incorporating underutilized computational resources through opportunistic computing. To achieve this, the framework integrates two well-established systems: Apache Spark for distributed data processing and HTCondor for managing non-dedicated, opportunistic resources.

The choice of Apache Spark is motivated by its suitability for large-scale, iterative data workloads and its robust support for in-memory computing. A more detailed overview of Spark advantages and role in Big Data environments is provided in Section 2.1.4. Within Spark deployment options, the standalone mode was chosen over YARN or Kubernetes for its

simplicity and minimal dependencies. This mode also allows direct control over scheduling logic, which proved essential for integrating opportunistic scheduling mechanisms into the Spark scheduler.

HTCondor was selected as the underlying resource manager for opportunistic nodes. Its built-in capabilities for handling ephemeral and preemptible resources made it well-suited for this role. The rationale behind this choice and an overview of opportunistic computing are discussed in Section 2.2.

Docker containers were used to ensure consistent runtime environments across heterogeneous machines. Each Spark executor runs within a preconfigured Docker container, which includes the necessary binaries, configurations, and dependencies required for execution. Containers offer lightweight isolation and portability, which are crucial in opportunistic environments where hardware and software heterogeneity is common. In particular, features such as control groups (cgroups) allow fine-grained resource constraints to be enforced, ensuring fair and efficient use of CPU and memory across all nodes.

HTCondor has supported the Docker universe since version 8.3.6, released in 2015[83], providing native integration with Docker containers for job execution. Under this execution environment, HTCondor launches jobs inside Docker containers by specifying the required image in the job submit file. The scheduler ensures that only nodes capable of running Docker-based jobs are selected for execution. When a job is matched to a node, HTCondor invokes Docker to start the container with the defined resource limits and environmental parameters. Once inside the container, the job behaves as it would in a standard universe, but benefits from the encapsulated and reproducible environment defined by the Docker image.

This capability is particularly valuable in BigOPERA hybrid architecture, as it guarantees consistent execution behavior across both dedicated and opportunistic nodes, regardless of variations in host system configurations.

This hybrid design pairing Spark with HTCondor and encapsulating executors in Docker containers, allows BigOPERA to elastically extend beyond its dedicated infrastructure, harnessing idle machines to improve resource efficiency and fault tolerance.

3.1.2 Networking and Connectivity

To ensure that opportunistic nodes could communicate effectively with the Spark Master and HTCondor Central Manager, a minimal networking setup was employed. In this thesis, the nodes were deployed on separate subnets but within the same virtual network, enabling direct

connectivity without requiring NAT traversal or tunneling. However, BigOPERA does not impose strict network topology constraints. As long as basic IP-level connectivity is maintained, opportunistic nodes can participate in the framework from diverse network domains, including remote VLANs or even over VPN links. This flexibility allows organizations to repurpose idle machines across departmental boundaries or geographic locations, as long as those nodes can reach the Spark Master and receive jobs from the HTCondor scheduler. This design choice aligns with BigOPERA's goal of maximizing resource reuse with minimal infrastructural overhead.

3.2 Opportunistic Executor Scheduling in BigOPERA

To enable opportunistic scheduling, BigOPERA modifies the standard Apache Spark standalone scheduler to prioritize dedicated resources while opportunistically utilizing non-dedicated nodes. This is achieved through the customized scheduling logic in the `scheduleExecutorsOnWorkers` function. This function assigns Spark executors across a mix of dedicated and non-dedicated worker nodes by dynamically evaluating each node's capacity, availability, and classification (dedicated or opportunistic). Nodes marked as dedicated are prioritized by pre-sorting the array of available workers, ensuring opportunistic nodes are considered only when dedicated ones cannot satisfy resource demands.

The function ensures that:

- Each executor receives at least the minimum required cores, memory, and any additional resource requirements (e.g., GPUs).
- Opportunistic nodes are used only if dedicated nodes are fully utilized.
- The executor limit per application is not exceeded.

An executor can only be launched on an opportunistic worker if it satisfies all resource checks and has not exceeded the configured limits. Opportunistic workers may run multiple executors depending on the configuration, especially if `coresPerExecutor` is not defined (implying one executor per worker).

The following pseudocode summarizes the key steps:

Algorithm 1 Opportunistic Executor Scheduling Algorithm

```
1:  Input: List of usable workers W, Application info A
2:  Sort W with dedicated nodes first
3:  Initialize assigned cores and executors arrays
4:  Compute total available cores to assign
5:  while there exist free workers w ∈ W such that w can launch an executor do
6:      for each w in free workers do
7:          while w can launch executor and cores remain do
8:              Reserve minimum cores
9:              Increment executor count on w
10:             if spreadOutApps = true then break
11:             end if
12:         end while
13:     end for
14:     Update free workers set
15: end while
16: Return: Array of assigned cores per worker
```

The opportunistic executor scheduling algorithm in BigOPERA introduces a two-tiered allocation model that prioritizes dedicated nodes while opportunistically leveraging non-dedicated (opportunistic) resources. At each scheduling round, the list of available workers is sorted to place dedicated nodes first, ensuring that these stable resources are filled before considering less reliable ones. The scheduler iterates over this list and attempts to launch executors on each node that meets the required resource constraints (e.g., minimum CPU cores and memory). If sufficient resources are available, the algorithm reserves cores and increments the executor count on that node. This process repeats until all available executors have been assigned or no more nodes can satisfy the requirements.

A key element of the algorithm is the `spreadOutApps=true` setting, which alters the executor placement strategy across the cluster. By limiting the scheduler to assign only one executor per node, this option promotes a more even distribution of workload. As a result, the system becomes more resilient to individual node failures since the loss of a single machine affects fewer executors and minimizes disruption to overall execution. Avoiding the concentration of executors on a small subset of nodes also improves load distribution, which is particularly beneficial in environments with heterogeneous or volatile resources. As a result, BigOPERA can scale more effectively and maintain stable performance even as node availability changes over time.

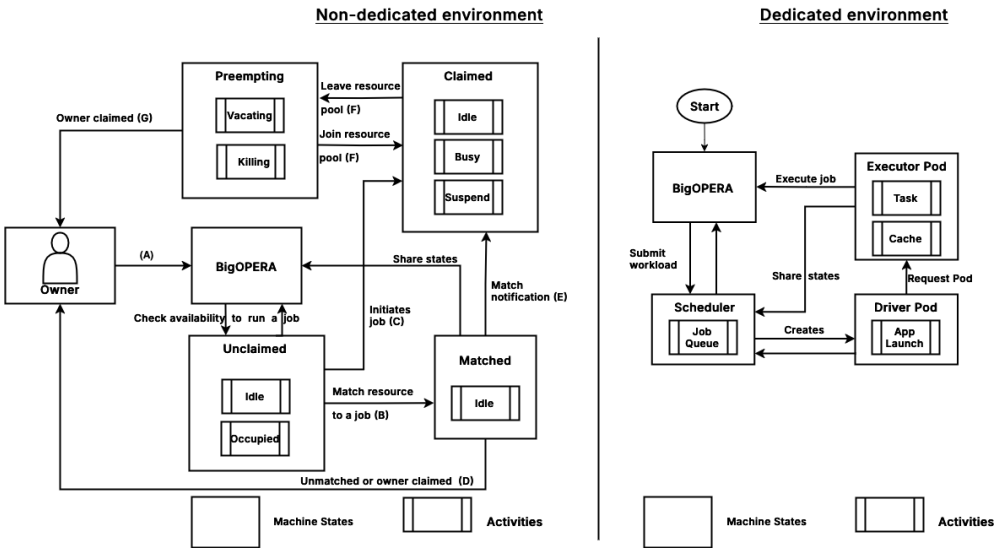


Figure 3.2: BigOPERA environments machine states and activities (Error states are omitted).

This algorithm ensures dedicated resources are always used first, with opportunistic resources dynamically activated based on current load and resource availability. Opportunistic workers are subject to stricter preemption and may lose executors if reclaimed by their original owners, which aligns with the machine state transitions described in Section 3.2. By integrating this scheduling logic, BigOPERA achieves a balanced trade-off between performance and cost, dynamically scaling execution across a heterogeneous resource pool while maintaining system robustness.

3.3 BigOPERA machine status

BigOPERA assigns machine states based on the availability of resources and their readiness to execute tasks. The system operates through six primary states, each representing a specific phase in a node lifecycle within the BigOPERA framework. These states are designed to ensure efficient resource utilization and task scheduling, even in dynamic and heterogeneous environments. The machine states, shown in the Figure 3.2, and their transitions are as follows:

- Owner indicates that the owner is using the node (e.g., running local tasks or processes); hence, it is not available to run a non-dedicated job.

- UNCLAIMED indicates that the node is idle and can run a non-dedicated job, yet it currently does not run any.
- MATCHED indicates that the node can run a non-dedicated job, and BigOPERA allocated it in the resource pool and matched it with a Driver Pod. That Driver Pod has not yet claimed this node by launching an Executor Pod. In this state, the machine cannot launch a job in the Executor Pod.
- CLAIMED indicates that the node can run a non-dedicated job, and BigOPERA assigned it with an Executor Pod, i.e., active to utilize this node.
- PREEMPTING: The node is in the process of being reclaimed by its owner, and the current task is being preempted.
- ERROR: The node has encountered an unrecoverable issue (e.g., hardware failure, network disconnection, or software crash) and cannot participate in task execution until the issue is resolved.

Another state not mentioned in Figure 3.2 is DRAINED, in which the node does not respond to the owner or BigOPERA. The ERROR state is also not depicted in the figure for simplicity, but it plays a critical role in handling failures and ensuring system robustness.

The state transitions are:

A The node switches from OWNER to UNCLAIMED whenever the START expression no longer locally evaluates to FALSE. This indicates that the node can run a BigOPERA job.

B The transition from UNCLAIMED to MATCHED happens whenever the DriverPod matches this resource with a BigOPERA job.

C The transition from UNCLAIMED directly to CLAIMED also happens if the DriverPod matches this resource with a BigOPERA job. In this case, the ExecutorPod receives the match and initiates the claiming protocol with the node before the DriverPod receives the match notification from BigOPERA DriverPod.

D The node moves from MATCHED to OWNER if the START expression locally evaluates to FALSE or the MATCH_TIMEOUT timer expires. This timeout ensures that if a node is matched with a given ExecutorPod but that Executor Pod does not contact the

DriverPod to claim it, the node will give up on the match and become available to be matched again. In this case, since the START expression does not locally evaluate to FALSE, the node will immediately enter the Unclaimed state again (via transition A). The node might also go from MATCHED to OWNER if the ExecutorPod attempts to perform the claiming protocol but encounters an error.

E The transition from MATCHED to CLAIMED occurs when the ExecutorPod completes the claiming protocol with the DriverPod.

F If the resource were matched to a job with a higher priority, it would move from PREEMPTING back to CLAIMED.

G The resource would move from PREEMPTING to OWNER if the PREEMPT expression had evaluated to TRUE or if the START expression locally evaluated to FALSE when the DriverPod had finished evicting whatever job it was running when it entered the PREEMPTING state.

Summary:

- Transitions are event-driven: Each state transition is triggered by specific events, such as task completion, owner reclaim, or task assignment.
- Timeout Mechanism: If a task does not start within a specified time after being matched, the node returns to the UNCLAIMED state to avoid resource wastage.
- Preemption Handling: The PREEMPTING state ensures that tasks can be gracefully interrupted if the owner reclaims the node, allowing for efficient resource sharing without disrupting ongoing operations.

3.4 BigOPERA Algorithms

The BigOPERA framework relies on three core algorithms to manage resource allocation, task execution, and fault tolerance. These algorithms work together to efficiently utilize both dedicated and non-dedicated (opportunistic) resources while maintaining system reliability and performance. This section discusses BigOPERA algorithms by separating distinct processes:

1. initialization and node state monitoring

Algorithm 2 BigOPERA: Initialization and State Monitoring

```
1: States: OWNER, UNCLAIMED, MATCHED, CLAIMED, PREEMPTING, ERROR
2: Initialize resource state as OWNER
3: while [Periodically check when the node becomes idle] True do
4:     if START local function evaluates to false then
5:         Transition to UNCLAIMED
6:     end if
7:     if error detected in local function then
8:         Transition to ERROR
9:     end if
10: end while
```

2. task matching and execution

3. reclaim management and error handling

Each algorithm reflects a critical operational phase within the BigOPERA runtime system.

3.4.1 Initialization and Node State Monitoring

The first phase of the algorithm (Algorithm 2) is responsible for initializing the resource state and continuously monitoring the machine availability and health. At startup, each node is assigned the OWNER state, indicating that it is actively used by its local owner and thus unavailable for opportunistic use. The algorithm periodically evaluates whether the node becomes idle (i.e., the START expression evaluates to `true`), in which case it transitions to the UNCLAIMED state and becomes available to execute BigOPERA workloads. Additionally, the algorithm includes a check for hardware or software faults; if an error is detected, the node transitions to the ERROR state. This monitoring process ensures that only healthy and idle nodes are considered for scheduling, contributing to the robustness of the system.

3.4.2 Task Matching and Execution

The second phase (Algorithm 3) governs the process of task assignment and execution. When a node is in the UNCLAIMED state, the BigOPERA scheduler evaluates whether it meets the requirements of any pending Spark tasks. If so, the node is matched to a task and enters the MATCHED state. A timeout mechanism ensures that the node will not remain indefinitely in this

state if the executor fails to initiate the claiming protocol; in that case, it reverts to UNCLAIMED, releasing any reserved resources.

If the executor successfully begins the claiming process, the node transitions to the CLAIMED state and starts executing the task using the reserved resources. This process enables flexible yet controlled task dispatching in a heterogeneous cluster, allowing opportunistic nodes to be used efficiently without sacrificing execution integrity or causing unnecessary delays.

Algorithm 3 BigOPERA: Task Matching and Execution

```
1:  while node is active do
2:      if state is UNCLAIMED then
3:          Scheduler Role: Check if resources meet the requirements of any pending tasks
4:          if sufficient resources are available for a task then
5:              Scheduler assigns task to resources
6:              Transition to MATCHED
7:              Reserve resources for the assigned task
8:          end if
9:      end if
10:     if state is MATCHED and task is ready to start then
11:         if owner requests reclaim of node then
12:             Transition to OWNER
13:         end if
14:         if timeout occurs before the task starts then
15:             Transition to UNCLAIMED
16:             Release reserved resources
17:         else if task is ready to start then
18:             Transition to CLAIMED
19:             Start executing the task using reserved resources
20:         end if
21:     end if
22: end while
```

3.4.3 Reclaim Management and Error Handling

The third algorithm (Algorithm 4) handles resource reclaiming and error recovery. If the owner of a node reclaims it while in the CLAIMED state, the system transitions to the PREEMPTING state. During this phase, the current task is either interrupted or allowed to complete, and the system notifies the Spark master that the node will no longer be available. Resources are then freed, and the node returns to the UNCLAIMED state, ready for reuse if it becomes

available again. In the case of critical failures (e.g., hardware faults, lost connectivity), the node enters the ERROR state. The system then periodically attempts to recover it. Upon successful recovery, the node transitions back to UNCLAIMED and re-enters the scheduling pool. These mechanisms ensure system resilience and fault tolerance, both of which are crucial for operating in volatile environments like those involving non-dedicated machines. Together, these algorithms establish the operational core of BigOPERA runtime behavior, enabling it to dynamically adapt to changing resource availability, respond gracefully to failures, and maintain high throughput across diverse infrastructure settings. Following is a simplified

Algorithm 4 BigOPERA: Reclaim Management and Error Handling

```
1:  if state is CLAIMED then
2:      Check if the owner requests the node back:
3:      if owner requests reclaim of resources during task execution then
4:          Transition to PREEMPTING
5:          Notify Spark master of preemption
6:          Stop accepting new tasks
7:          Interrupt current task (if needed) or allow the task to finish
8:          Save task progress if applicable
9:          Free up resources
10:         Transition to UNCLAIMED
11:         Notify the master of resource availability
12:     end if
13:     if task completes (without preemption) then
14:         Free up resources
15:         Transition to UNCLAIMED
16:     end if
17: end if
18: if state is PREEMPTING then
19:     Wait for the current task to finish (if any)
20:     Free up resources
21:     Transition to UNCLAIMED
22:     Notify Spark master of resource availability
23: end if
24: if state is ERROR then
25:     Attempt to recover from error
26:     Transition to UNCLAIMED if recovery successful
27: end if
```

flowchart of the state transitions in BigOPERA:

- Owner → Unclaimed: Node becomes idle and available for BigOPERA tasks.
- Unclaimed → Matched: Resources are matched to a task.
- Matched → Claimed: Task starts execution.
- Claimed → Preempting: Owner reclaims the node, and the task is preempted.
- Preempting → Unclaimed: Resources are freed after preemption.
- Error → Unclaimed: Node recovers from an error and becomes available again.

CHAPTER 4

PERFORMANCE EVALUATION

This chapter provides the performance evaluation of the BigOPERA framework when executing distributed data processing workloads using Apache Spark. The evaluation begins with a description of the experimental environment, including the hybrid cluster deployment configured on the Google Cloud Platform (GCP) and the resource profiles used for both dedicated and opportunistic nodes. It then introduces the benchmark suite employed for testing, outlining the selected applications, the rationale for their inclusion, and the data sizes used in each case. Following this, the chapter analyzes the system execution under various conditions. The results are used to assess the effectiveness, scalability, and resilience of the BigOPERA architecture compared to a fully dedicated setup, highlighting its suitability for real-world, large-scale data analytics workflows.

4.1 Experimental Setup

The evaluation of BigOPERA Spark was conducted by comparing the performance of a dedicated cluster with a hybrid cluster composed of dedicated and opportunistic resources, both running on GCP. The dedicated cluster consisted of four worker nodes type *n1-standard-2*, with two vCPUs and 7.5 GB of memory plus an additional node for the control plane running on Kubernetes. In contrast, the hybrid setup leveraged a combination of that dedicated cluster alongside four additional *e2-medium* with 2vCPU, 4 GB of memory nodes, and a *e2-medium* HTCondor manager node (not running in Kubernetes), a 1:1 opportunistic to dedicated node ratio was created. We set a 2 GB limit per executor for the memory difference between instances. The experimental settings were primarily based on the default configurations of

Intel HiBench. This ensures that the results are comparable to other studies using the same benchmark. Additionally, specific parameters were adjusted for our particular environment, as recommended in the official documentation [84]. The number of executors was set to four/eight, and two cores, ensuring that each worker fully utilized its two cores for efficient parallel execution. Memory allocation was set to 2 GB, limiting executor memory to accommodate differences in available memory across instance types. Parallelism settings were configured to eight/sixteen for *map* and *shuffle*, as clusters require sufficiently high parallelism levels to maximize utilization. According to Spark documentation, setting these values to twice the number of cores is recommended [85].

These configurations align with Apache guidelines, strengthening the validity of the experimental setup.

4.2 Testbed and Dataset Description

methodology Our workloads focused on several representative MapReduce applications: Wordcount, Sort, Terasort and SparkPi. To automate the configuration, execution and results gathering, we used Intel's HiBench Suite[1].

HiBench [86] is an extensive benchmark suite for BigData platforms such as Hadoop, Spark, Storm, etc. It includes a total of 29 workloads divided into six categories: Micro, machine learning, SQL, graph, web search, and streaming. The applications used for this evaluation were:

- 1. WordCount: This application counts how often each word appears within the input data generated using RandomTextWriter.
- 2. Sort: This application sorts its text input data, generated using Random- TextWriter.
- 3. TeraSort: a standard map/reduce sort, except for the use of a custom partitioner.
- 4. SparkPi: Computes an approximation to π by throwing darts at a circle. It is not included on HiBench but is available as an example on Spark.

Table 4.1 shows the input data sizes used for the HiBench benchmarks. In the case of SparkPi, we started with 200,000 partitions and increased them by 100,000 up to 800,000.

[1] All the code necessary for running these benchmarks has been ported to Python 3, and it is available at: https://github.com/kadern0/HiBench.

Table 4.1: Input Datasizes

	Tiny	**Large**	**Huge**	**Gigantic**	**Bigdata**	**2-bigdata**
Wordcount	*NU*	388.4 MB	3.79 GB	37.9 GB	89.6 GB	379.3 GB
Sort	*NU*	38.8 MB	388.4 MB	3.79 GB	35.5 GB	71.1 GB
TeraSort	3.0 MB	305.18 MB	2.98 GB	29.8 GB	298.02 GB	*NU*

Each benchmark scenario was executed three times to ensure consistency and account for potential variability. The reported results represent the average values across these three runs. We simulated ideal conditions for these tests, where no opportunistic node was reclaimed during task execution. In the following section, we deeply analyze the performance impact of node failures.

4.3 Results and Discussion

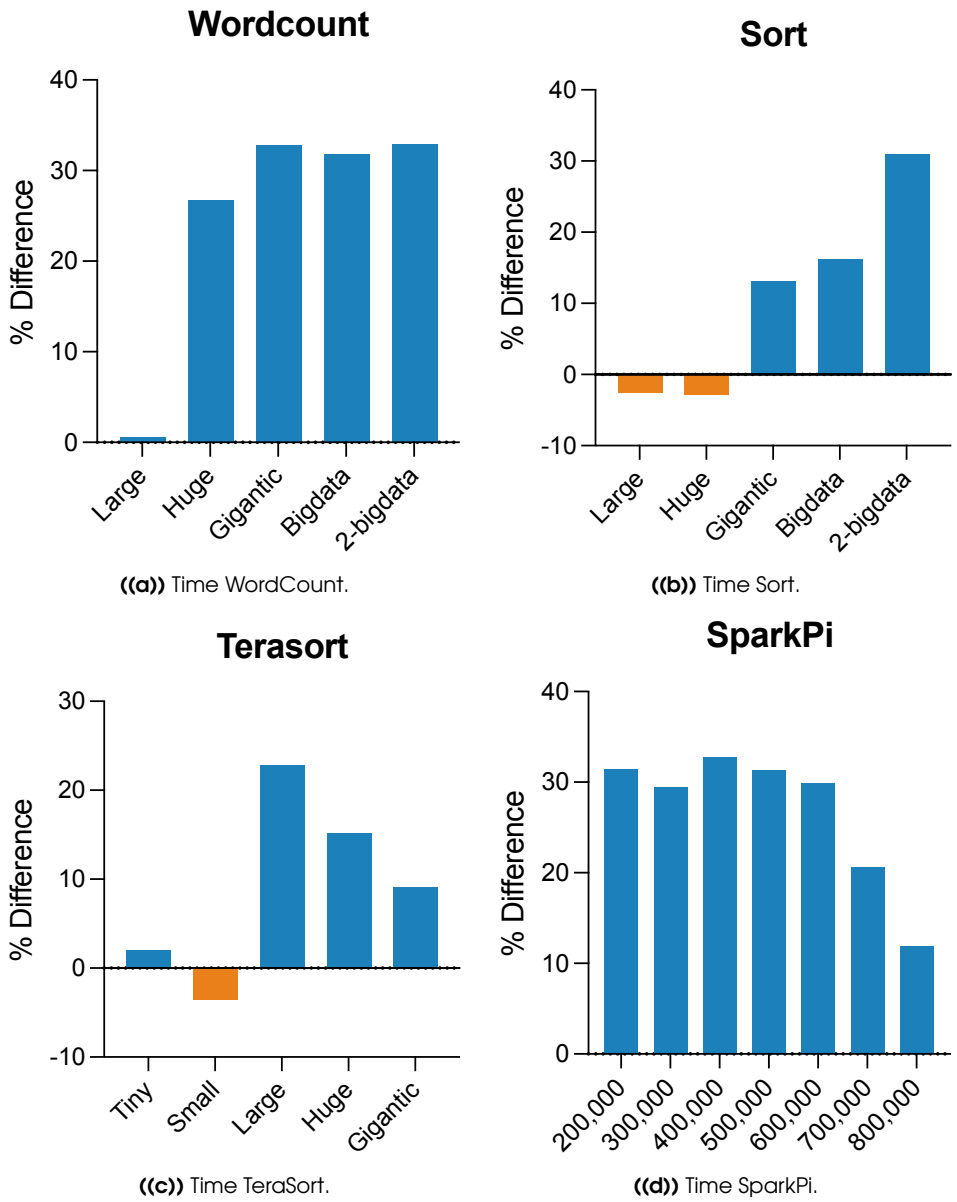
This section presents an analysis of the system performance, followed by an examination of the fault tolerance. The performance analysis focuses on execution times using the benchmarks described in the previous section, providing insights into the system's efficiency and scalability. Subsequently, we investigate the system's fault tolerance, evaluating its ability to maintain functionality and recover from node losses.

4.3.1 Performance

Figure 4.1 shows the percentage difference in execution time for each of the benchmarks across different input sizes, comparing the dedicated-only Spark setup with the hybrid BigOPERA configuration. Each subfigure corresponds to a specific benchmark: WordCount, Sort, TeraSort, and SparkPi.

In Figure 4.1(a), the difference in execution times for the Wordcount benchmark is illustrated, with BigOPERA Spark showing a significant improvement over Apache Spark. On average, BigOPERA was approximately 35% faster, consistently outperforming Apache Spark alone.

In the Sort benchmark, Figure 4.1(b), we observed varying degrees of improvement with BigOPERA over Apache Spark. For datasets smaller than 400 MB, BigOPERA was slightly slower due to the scheduling overhead introduced by managing both dedicated and opportunistic resources, which outweighs the performance gains for small workloads. However, for



((a)) Time WordCount.

((b)) Time Sort.

((c)) Time TeraSort.

((d)) Time SparkPi.

Figure 4.1: Difference in execution times between Apache Spark and BigOPERA Spark (the higher, the better) for different benchmarks.

larger datasets, BigOPERA exhibited execution times between 10% to 30% faster than Apache Spark, with the performance advantage increasing with dataset size.

Similarly, in the Terasort benchmark, Figure 4.1(c), BigOPERA Spark exhibited notable performance gains over Apache Spark. For datasets smaller than 300 MB, the execution times were comparable. For larger datasets, BigOPERA showed an improvement of 10% to 25% compared to Apache Spark across multiple trials. However, the degree of improvement diminished with increasing dataset size (further explained in 4.3.2).

In the SparkPi benchmark, Figure 4.1(d), BigOPERA showcased substantial improvement in execution time over Apache Spark. On average, BigOPERA completed the computation approximately 30% faster. This improvement can be attributed to the framework ability to dynamically leverage additional opportunistic resources during execution, enhancing parallelism. Similar to the Terasort benchmark, the performance gain decreased with larger workloads.

4.3.2 Throughput Degradation at Scale

During the execution of the TeraSort and SparkPi benchmarks, we observed a decline in performance improvement when processing large input sizes. This behavior was consistent across both dedicated and hybrid configurations, as shown in Figure 4.2 for the TeraSort benchmark, suggesting a combination of resource saturation and system-level bottlenecks.

In the dedicated-only setup, performance plateaued as input sizes increased beyond moderate scales. While the initial growth in throughput was proportional to data size, gains diminished with larger inputs. This effect can be attributed to CPU and memory saturation and increased I/O overhead from disk spilling. In shuffle-intensive workloads such as TeraSort, the volume of intermediate data grew substantially, resulting in heightened disk and network I/O demands. Similarly, the iterative and partition-heavy nature of SparkPi added to the overall computational burden.

In the hybrid configuration, which incorporated opportunistic `e2-medium` instances, a comparable pattern emerged. Although these additional nodes initially contributed to improved throughput between 3 GB and 30 GB of input data, performance peaked at around 30 GB and then declined. The opportunistic resources introduced greater heterogeneity in execution, and their non-deterministic behavior under sustained load—such as CPU throttling or scheduling delays—limited further scalability.

Importantly, `e2-medium` instances in Google Cloud support short-term CPU bursting, allowing temporary performance boosts for up to 120 seconds. While this feature can absorb

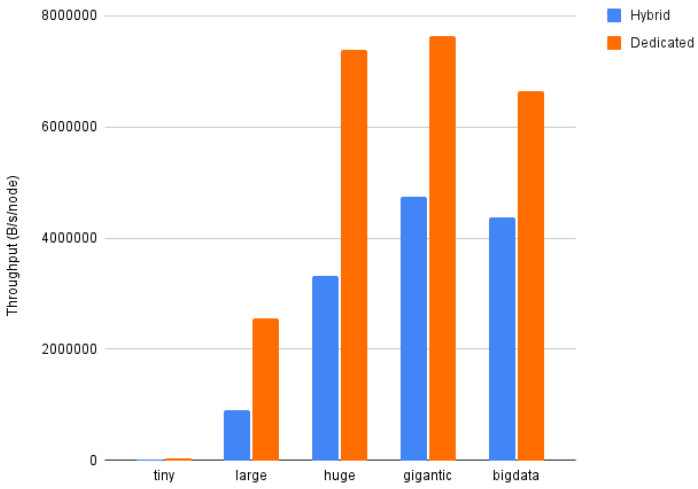


Figure 4.2: Node throughput reported by HiBench expressed in bytes per second per node for Terasort.

short-lived peaks in demand, it is insufficient for prolonged high-throughput workloads, such as those observed during extended shuffle or compute-bound stages in our benchmarks. As a result, after the bursting window is exceeded, these instances may exhibit reduced performance, contributing to the observed decline in throughput beyond 30 GB of input data [87, 88].

4.3.3 Fault tolerance

Given the ephemeral nature of opportunistic resources, where nodes can be reclaimed, shut down, or disconnected from the network, the system's capacity to cope with node losses is critical to evaluate.

Initial versions of Apache Spark encountered significant challenges when dealing with tasks that required even moderately transient resources. This issue was particularly evident in scenarios where Spark struggled to complete tasks, leading to application failures that were difficult to diagnose and resolve. The study by Bowen Yu et al. noted that early versions of Spark did not handle tasks efficiently when resources were short-lived, highlighting a critical limitation in its original design [89].

The works on [79] [80] [40] enhanced Spark resiliency to node loss, mainly focused on improving reliability and performance when using spot instances on cloud environments.

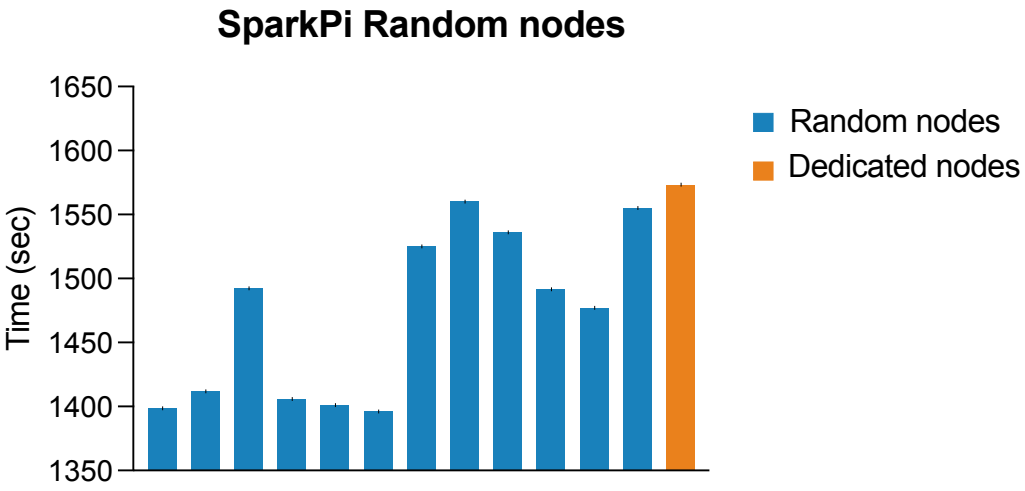


Figure 4.3: SparkPi on random nodes

Nevertheless, these improvements also made it possible to use opportunistic computing to run Spark jobs.

To test BigOPERA node disruption tolerance, we measured the time it took to run the SparkPi application with 800,000 partitions using the same setup as in Figure 4.1(d), although this time, we introduced node disruptions to verify the impact of such outages on the overall execution time. We used the time it took to run the benchmark on the dedicated cluster as base measurement and then stopped the opportunistic nodes randomly, thus simulating node failures in a real-life scenario. We assumed node outages were mutually independent and ran a script that randomly stopped and started the HTCondor tasks.

Figure 4.3 shows that the hybrid cluster was faster than the dedicated cluster. Despite the expected variability in execution times, the hybrid cluster was consistently quicker, demonstrating that the opportunistic nodes, despite being randomly stalled, could provide a performance advantage.

These findings suggest that the ability of the opportunistic cluster to utilize idle resources effectively outweighed the potential disruptions caused by the random node loss.

This chapter partially reproduces the publication:

- Pablo V. Caderno, Feras Awaysheh, José C. Cabaleiro, and Tomás F. Pena. "BigOPERA: An OPportunistic and Elastic Resource Allocation for big data frameworks".

In: *Cluster Computing* (2025). Affiliations: Caderno, Pablo Vazquez (CiTIUS research center, University of Santiago de Compostela, Spain), Awaysheh, Feras M. (Department of Computing Science, ADSLab, Umeå University, Umeå, Sweden), José C. Cabaleiro (CiTIUS research center, University of Santiago de Compostela, Spain), Tomás F. Pena (CiTIUS research center, University of Santiago de Compostela, Spain). DOI: 10.1007/s10586-025-05274-4

CHAPTER 5

USE CASE: BIGOPERA IN GENOME ANALYSIS

This chapter presents a practical application of the BigOPERA framework in the domain of genome analysis. In this deployment, we refer to the customized version as OPERA-gSAM. The key distinction between the generic BigOPERA and OPERA-gSAM lies in the integration of Google Cloud Storage (GCS) connector. Since the experimental data for this use case was stored in GCS instead of HDFS, it was necessary to build a new Docker container incorporating the GCS connector. This new version of the container, specifically tailored for this workflow, was tagged as `OPERA-gsam`. This case study has been published as [91].

The following sections describe the underlying challenges of UMI processing, the architecture of the OPERA-gSAM deployment, and the experimental evaluation demonstrating its benefits over traditional execution models.

5.1 Introduction

In the last two decades, NGS technologies have revolutionized the field of genomics by enabling the comparison of DNA or RNA fragments across different biological conditions. These technologies rely on counting the number of "reads" mapping to specific genomic areas, and a crucial part of the sequencing workflow involves amplifying the number of original DNA/RNA molecules using Polymerase Chain Reaction (PCR). This amplification step is necessary to generate enough material for sequencing. However, it also introduces duplicates,

which must be identified and discarded to accurately quantify the number of unique molecules in the original sample. Unique molecular identifiers (UMIs) facilitate this by allowing for the identification and removal of PCR duplicates during data processing [92]. UMIs are random nucleotide barcodes added to DNA molecules before amplification, enhancing the accuracy of quantification [93].

Recent research has highlighted that a significant issue in UMIs performance relates to assigning reads to genes and counting molecule stages. Such stages in the UMI pipeline are computing-intensive (mainly memory), like the Spliced Transcripts Alignment to a Reference (STAR). STAR is an aligner designed to address many of the challenges of RNA-seq data mapping using a strategy to account for spliced alignments [94]. In particular, many of these tasks are data I/O bound, and the nature of the framework, deployment model, and data pipeline attached to the storage will tremendously affect the performance. Running on multi-thread processors and SSD memory will give a substantially quicker performance. Nevertheless, such traditional approaches will tremendously boost the required computational resources.

Advancement of data-intensive bioinformatics processing and Big Data (BD) frameworks like Apache Spark can aid in improving the performance of a broad scope of applications [95]. In this context, we explore the application of the BigOPERA framework to genome analysis, specifically targeting UMI-based single-cell RNA-seq pipelines. This deployment, named OPERA-gSAM, demonstrates how BigOPERA can be effectively leveraged to enhance the scalability and efficiency of three key stages in the UMI processing pipeline: (i) sorting, (ii) index-sorted alignment of SAM files, and (iii) counting unique molecules by identifying and correcting UMIs aligned to each gene. OPERA-gSAM can employ the unutilized (idle) resources for exploiting the full cluster capabilities and better return on infrastructure investments. In this work, we argue that OPERA-gSAM addresses the scalability and agility of standard BD bioinformatics analysis tools (e.g., UMI) and can tackle conceptual developments in the applications of parallel distributed computing to genomics.

5.2 Background

This section provides the necessary background to understand the role and computational demands of Unique Molecular Identifiers (UMIs) in modern bioinformatics pipelines and the computational challenges posed by large sequencing datasets.

5.2.1 UMI in bioinformatic pipelines

The current bioinformatic pipelines using UMIs include the following two steps: first, aligning reads to a reference genome or transcriptome, to assign genomic coordinates or genes to each read. Second, counting the number of unique molecules corresponding to each gene or genomic locus, and in the case of single-cell experiments, per cell. UMI-tools is a software suite written in python [96] to quantify UMIs accurately considering the possible errors introduced within the UMI sequenced. For instance, nucleotide substitutions during PCR or nucleotide miscalling during sequencing. Preparing the SAM file for counting, which includes sorting and indexing with the SAM tools package [97] and counting the UMIs using the UMI-tools count function, are computing-intensive tasks. The SAM files can also be massive (see Figure 5.1), varying between one GB and hundreds of GB depending on the method, sample species, and sequencing depth used. Moreover, each experiment usually includes multiple samples, so sorting and counting needs to be done repeatedly for each sample. Thus, improving the scalability and efficiency of these steps will significantly impact the overall processing time of UMI-based NGS sequencing.

Counting reads with unique genome coordinates and UMIs is crucial to estimate the number of DNA/RNA molecules across different biological samples. UMIs are most frequently used in single-cell RNA-seq [98, 99]. In this case, the source material is so small that a high number of PCR cycles are required, creating a large number of PCR duplicates that need to be discarded. However, UMIs can be applied to any sequencing method where detecting PCR duplicates using alignment coordinates alone is challenging. UMIs have been used for ChIP-seq [100], DNA-seq [101], single-cell ATAC-seq [102], spatial transcriptomics [103] and antibody repertoire sequencing [104].

5.2.2 Related work

Bioinformatics analyses have a long tradition of adopting technologies from the Big Data community. Such research aims to tackle the demand for scalable and efficient technologies to handle data-intensive and computing-intensive bioinformatics pipelines [105, 106, 107].

A recent study reported an optimized version of the sci-RNA-seq protocol with three rounds of split-pool for combinatorial indexing [108]. Investigating the impact of different data pipelines on performance, computational consumption, and consistency was the aim of [109].

[1] Adapted from Biorender.com

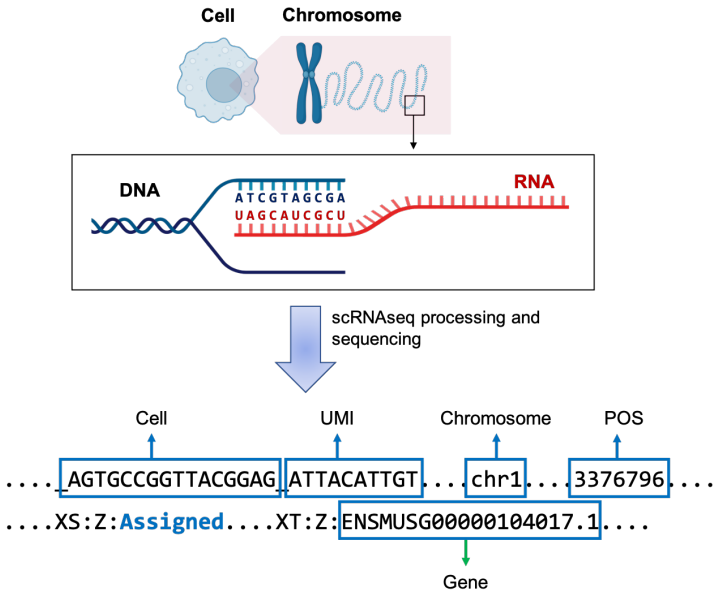


Figure 5.1: Schematic representation of the biological sample processing and SAM file format for data interpretation. [1]

This study encapsulated seven existing high-throughput scRNA-seq data processing pipelines with Nextflow, a general integrative workflow management framework. They evaluate eight public datasets generated from five high-throughput scRNA-seq platforms. Likewise, in work by José M Abuín et al. [110, 111], different processing models were conducted to accelerate the alignment stage. A similar effort to introduce a scalable and in-memory processing engine to process Whole-Genome-Sequencing in parallel [112].

In scRNA-seq computational tasks, big data tools like Spark and cloud computing can perform various tasks [113, 114], such as data pre-processing, quality control, normalization, dimensionality reduction, clustering, and visualization. For example, Spark can be used to perform pre-processing data tasks, like filtering and trimming reads, aligning reads to the reference genome, and quantifying gene expression levels. Cloud computing can store and manage scRNA-seq data and perform downstream analysis tasks. For instance, dimensionality reduction using principal component analysis (PCA) or t-distributed stochastic neighbor embedding (t-SNE), k-means or hierarchical clustering, and visualization using tools. Big data tools like

Spark and cloud computing can provide a scalable and efficient platform for processing and analyzing scRNA-seq data. They can help researchers to overcome the challenges associated with large-scale data processing and enable the discovery of new biological insights at the single-cell level.

Many recent studies utilize de-facto big data solutions like Apache Spark for medical Sequencing research [115]. Apache Spark and MPI (Message Passing Interface) are both distributed computing frameworks that can be used for processing and analyzing large-scale data. However, they have different use cases and impacts on the bio datasets. Apache Spark is a high-level BD processing framework designed for processing large-scale data, including metagenomics data. At the same time, MPI is a low-level message-passing interface used for parallel computing in HPC environments. Comparing both processing models was conducted [114] and the speed up reported in [116].

Against all previous publications, OPERA-gSAM provides scalable, agile, and easy-to-use Spark-based genome Sequence Alignment Map in the UMI tools. OPERA-gSAM utilizes recent advancements in the BD realm to improve processing UMI pipelines with I/O-intensive and computing-intensive stages with high performance while consuming less computational resources. Also, its novel architecture provides elastic resource allocation with an on-demand deployment model.

5.3 OPERA-gSAM Architecture

Figure 5.2 introduces the Bioinformatic steps for UMI-based single-cell RNA-seq pipelines and the associated OPERA-gSAM stages.

The output of the Illumina sequencer is a FASTQ file, a text-based format that stores the nucleotide sequence of each read detected by the sequencer and a quality score for each nucleotide. FASTQ files are first aligned to the reference genome of the matching species from the sample. The output file of this step is a SAM file [117], which is then assigned to the corresponding gene using a reference gene gtf file. The SAM file then has the nucleotide sequence of each read, the genome coordinates, which are defined as the chromosome number and the nucleotide location of the start of the read in that chromosome, and the corresponding gene for that locus (see Figure 5.1). SAM files can also be stored as BAM files, which is the binary version of a SAM file. Before counting the number of unique molecules corresponding to each gene or genomic locus, we need to sort and create an index of the SAM file. Finally, we

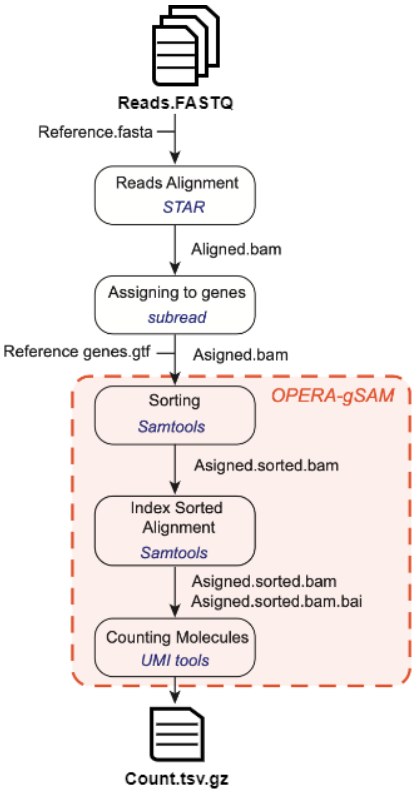


Figure 5.2: Bioinformatic steps for UMI-based single-cell RNA-seq pipelines.

will count the number of unique UMIs per gene per cell and store it in a .csv file. Figure 5.2 represents the UMI tool's main stages and the OPERA-gSAM contribution abstractly. The OPERA-gSAM framework combines sorting, indexing, and counting in one highly efficient step.

We used Apache Parquet instead of the *.csv* format to improve the sorting performance. Apache Parquet is an open-source, structured, column-oriented (aka columnar storage), compressed, binary file format for efficient storage and retrieval.. It offers better performance with compression support, which reduces the data size on the disk and, consequently, the I/O and CPU resources required to deserialize data.

Figure 5.3 visually represents Spark internal execution plan after the Catalyst optimizer run. Important to notice in that figure is the *Shuffle* operation (green rectangle), where nodes

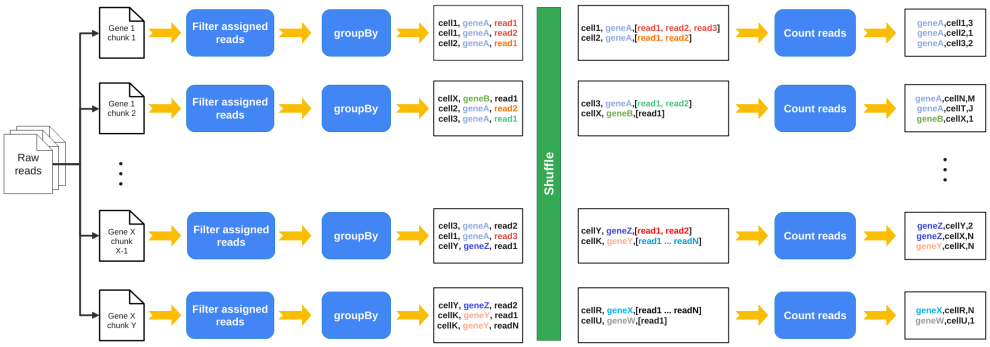


Figure 5.3: UMI counts Spark execution plan.

need to exchange data with each other. Shuffling is a time-consuming operation, so incurring on several shuffling tasks would have a detrimental impact on the overall performance of the code.

Algorithm 5 PySpark UMI Counting Algorithm

1: **Input:** Stored Parquet data $initialDataframe$
2: Initialize empty $tempDataframe$
3: **for** each row in $initialDataframe$ **do**
4: **if** $row[assigned]$ is not `NULL` **then** ▷ Filter unassigned reads
5: Append $RNAME, CELL, GENE, LIST(UMIs)$ to $tempDataframe$
6: **end if**
7: **end for**
8: **for** each row in $tempDataframe$ **do**
9: $readsDF \leftarrow$ aggregate by $gene, cell$ counting distinct $UMIs$
10: Write $readsDF$ to output
11: **end for**
12: **Return:** Processed UMI counts dataframe

Our code, described in Algorithm 5, loads the stored data in Parquet format on a DataFrame. This initial DataFrame is filtered by discarding all rows without *Assignment* information (the reads that are not mapped to any gene). We extract the UMI and the cell from the *QNAME* column on the mapped rows. We then group the reference sequence name (*RNAME*) along with the cell and the gene-tag (column *XT*) and aggregate all the UMIs that correspond to that combination of values. This new intermediate DataFrame contains all the raw UMIs, including repetitions and errors. The last processing step consists of removing duplicates and

discarding read errors. For that purpose, we developed a function that iterates through the UMIs list and calculates the Hamming distance between all the items, discarding those with a Hamming distance lower than two (the two UMIs have at least 2 different characters on the corresponding positions), and returning the number of unique error-free UMIs. This last DataFrame, shown in Figure 5.4, is saved on the disk.

```
+--------------------+----------------+-----+
|gene                |cell            |reads|
+--------------------+----------------+-----+
|ENSMUSG00000033793.11|AAAAACTTGGTGAGAG|1    |
|ENSMUSG00000033793.11|AAACCCATCCGCAAAT|2    |
|ENSMUSG00000033793.11|AAACGAAAGCTCCACG|3    |
|ENSMUSG00000033793.11|AAACGCTTCTAGCCTC|1    |
|ENSMUSG00000033793.11|AAAGAACCAACAACAA|1    |
|ENSMUSG00000033793.11|AAAGAACGTACCACGC|1    |
|ENSMUSG00000033793.11|AAAGACCAGCCATTTG|1    |
|ENSMUSG00000033793.11|AAAGGATGTATGTGTC|1    |
|ENSMUSG00000033793.11|AAAGGGCGTCCTACGG|2    |
|ENSMUSG00000033793.11|AAAGTCCGTCTAACTG|1    |
+--------------------+----------------+-----+
```

Figure 5.4: Sample output from the UMI counting step showing gene, cell, and read counts.

5.4 Experiment and Validation

To validate the OPERA-gSAM framework, we used data from a scRNA-seq of lungs from a metastatic breast cancer mouse model. Briefly, lungs from syngeneic triple-negative breast cancer mice were harvested and processed to prepare single-cell suspensions [118] that were subsequently loaded in the 10x Chromium System (10x Genomics). The RNA sequencing libraries were generated with the Single Cell 3' Library kit V3 from 10x Genomics and sequenced in an Illumina NovaSeq6000 sequencer.

Given that SAM files have columnar and csv tab-separated formats, we compared the performance of our code using csv and a columnar data format, i.e., Apache Parquet. According to the literature [119] and the initial tests performed within the local development cluster, the Apache Parquet format significantly outperforms the row-oriented file formats (csv) in our case. Local tests were carried out using a 14.2 Gb SAM file and using 1 to 4 executors with 2 CPU cores and 2 Gb of memory per executor. As shown in Figure 5.5 and Figure 5.6, Parquet yielded better performance in both sorting and UMI counting tasks.

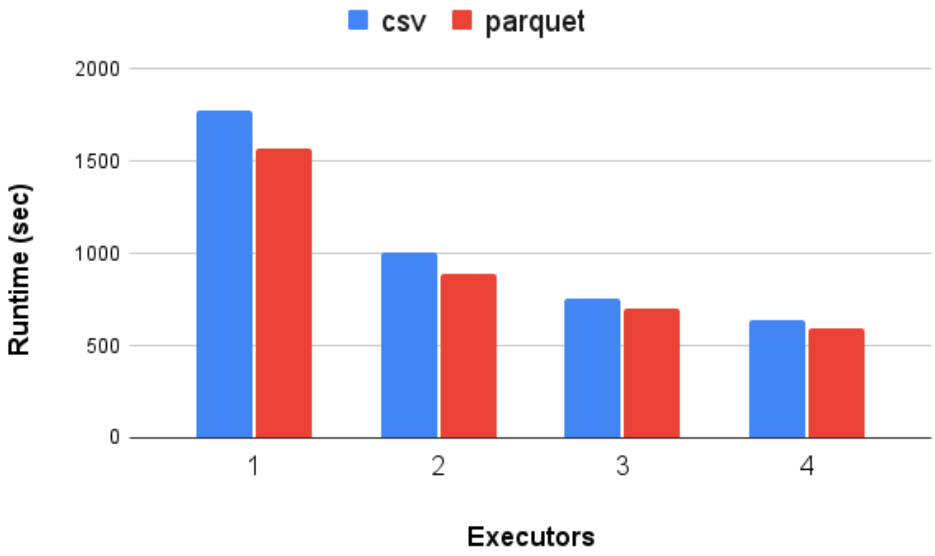


Figure 5.5: Local development cluster counting times

After reviewing the results on the local cluster, we chose Apache Parquet as the storage format for our experiment.

The OPERA-gSAM setup consisted of a dedicated cluster running on Kubernetes joint with additional non-Kubernetes nodes simulating an *opportunistic* environment. For the dedicated cluster, 4 worker nodes type *n1-standard-2*, with 2 vCPUs and 7.5 GB of memory (2 GB used per executor), were used. The hybrid setup added 4 additional nodes of type *e2-medium* with 2 vCPU and 4 GB of memory limited to 2 GB per executor.

The baseline metrics were obtained by running SAM tools and UMI-tools on a high-performance cluster with 56-core Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40 GHz processors. We set a maximum memory limit of 32 GB to ensure fair testing conditions and required 32 cores for each run.

The benchmarks performed compared the time it took to sort the SAM files and count the unique, error-corrected UMIs per gene, using three different input data sizes: 14.2 GB, 28.4 GB and 56.9 GB on both platforms.

Figure 5.7 shows that the HPC is faster than OPERA-gSAM at sorting the files, and while the performance gap continues to grow with larger input sizes, the rate at which it increases

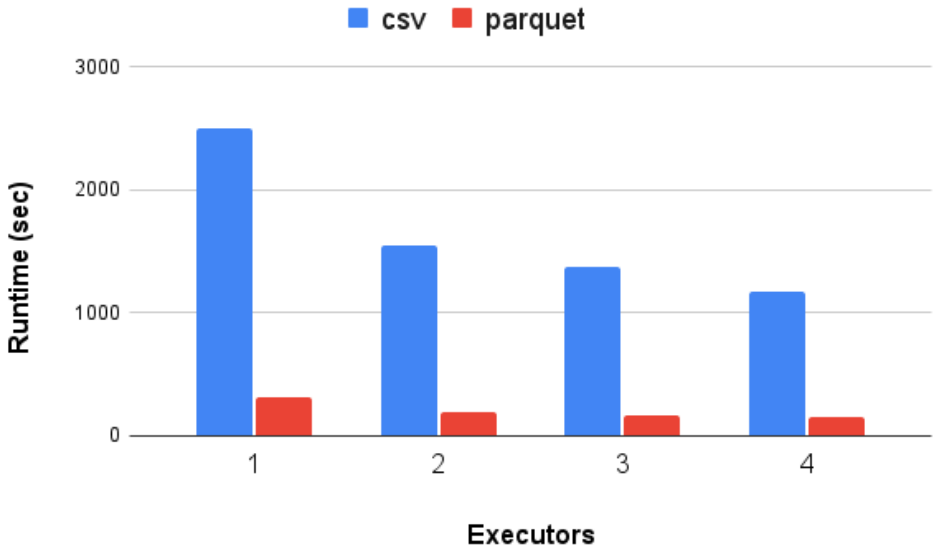


Figure 5.6: Local development cluster sorting times.

diminishes. Several factors likely cause this: the HPC node can fit in memory the whole file, or most of it, at once, thus minimizing disk I/O latency, compared to OPERA-gSAM, which uses half of the memory (32 Gb vs 16 Gb). Moreover, the HPC node reads the file from a local network share whilst OPERA-gSAM uses a cloud storage backend (in this case, Google Cloud Storage).

When we compared the time it took to count the UMIs, we performed two different measurements within the HPC. One of them used the unordered SAM file. For the other one, we sorted the SAM file and transformed it to its corresponding binary format, BAM, which increases the efficiency of the count operation. However, this conversion requires an additional step, indexing, before counting. Figure 5.8 shows that counting the UMIs directly on an unordered SAM file is quite inefficient. By converting to binary format, sorting and indexing the file, we obtain a speedup between 1.51x and 1.76x. OPERA-gSAM outperformed both approaches, resulting in speedups ranging between 1.7x and 2.3x from the optimized HPC approach.

This chapter partially reproduces the publication:

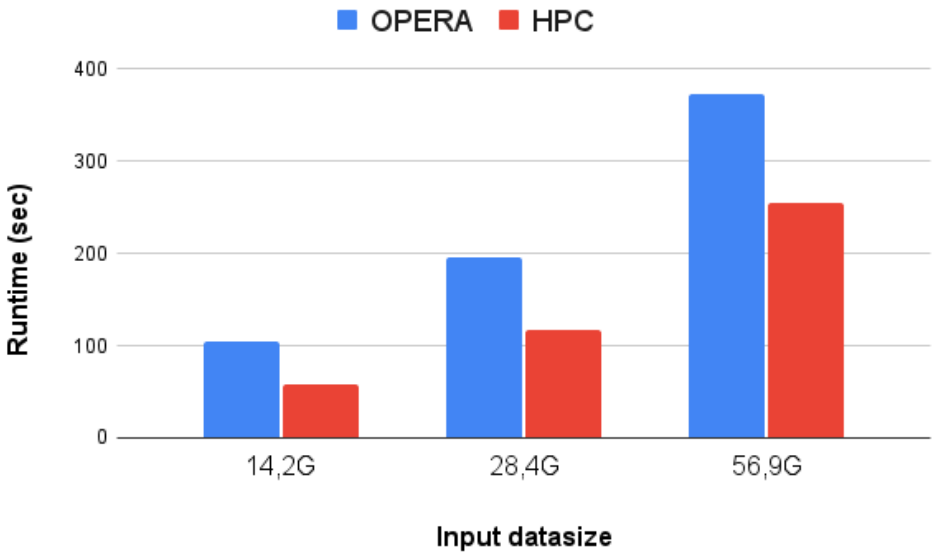


Figure 5.7: SAM sort times.

- Pablo Vazquez Caderno, Feras M. Awaysheh, Yolanda Colino-Sanguino, Laura Rodriguez De La Fuente, Fatima Valdes-Mora, Jose C. Cabaleiro, Tomas F. Pena, and David Gallego-Ortega. "OPERA-gSAM: Big Data Processing Framework for UMI Sequencing at High Scalability and Efficiency". In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*. Affiliations: Caderno, Pablo Vazquez (CiTIUS research center, University of Santiago de Compostela, Spain), Awaysheh, Feras M. (Institute of Computer Science, Delta Center University of Tartu, Tartu, Estonia), Colino-Sanguino, Yolanda (Cancer Epigenetic Biology and Therapeutics Laboratory, Children's Cancer Institute, Sydney, Australia School of Clinical Medicine, University of New South Wales, Sydney, Australia), Laura Rodriguez De La (Cancer Epigenetic Biology and Therapeutics Laboratory, Children's Cancer Institute, Sydney, Australia The Kinghorn Cancer Centre, Garvan Institute of Medical Research, Sydney, Australia St. Vincent's Clinical School, Faculty of Medicine, University of New South Wales, Sydney, Australia), Valdes-Mora, Fatima (Cancer Epigenetic Biology and Therapeutics Laboratory, Children's Cancer Institute, Sydney, Australia School of Clinical Medicine, University of New South Wales, Sydney, Australia),

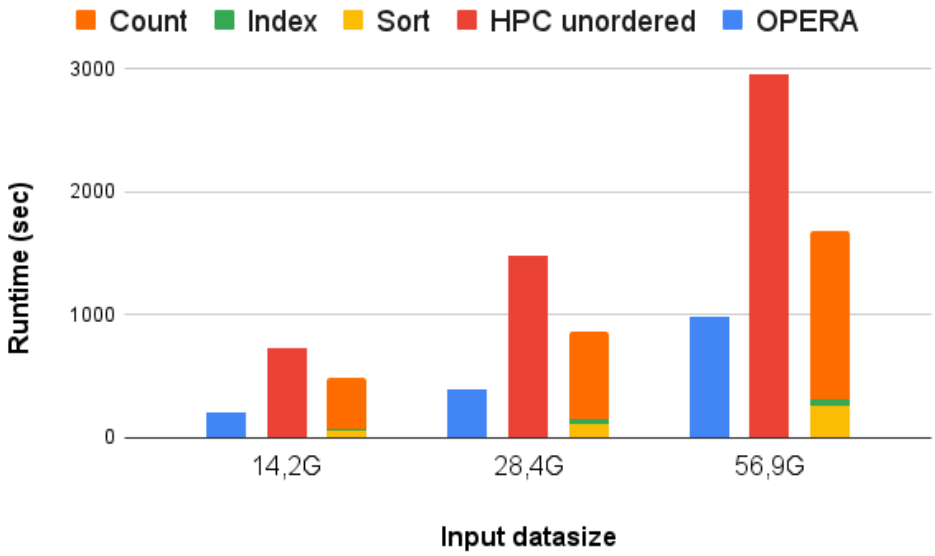


Figure 5.8: UMI count times.

Cabaleiro, Jose C. (CiTIUS research center, University of Santiago de Compostela, Spain), Pena, Tomas F. (CiTIUS research center, University of Santiago de Compostela, Spain), Gallego-Ortega, David (The Kinghorn Cancer Centre, Garvan Institute of Medical Research, Sydney, Australia St. Vincent's Clinical School, Faculty of Medicine, University of New South Wales, Sydney, Australia SC of Biomedical Engineering, FAC of Engineering and Information Technology, University of Technology, Sydney, Australia). 2023, pp. 160–167. DOI: 10.1109/CCGridW59191.2023.00038

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This thesis presented BigOPERA, a hybrid architecture designed to integrate dedicated and opportunistic resources into BD processing frameworks. It specifically focused on enhancing the elasticity, efficiency, and fault tolerance of Apache Spark workloads. The primary objective was to address the underutilization of computational resources and the limitations of static, dedicated-only infrastructures in the context of large-scale data processing.

To this end, the thesis proposed a two-tiered scheduling mechanism that allows Spark to prioritize dedicated nodes while opportunistically incorporating idle or ephemeral resources. This approach was implemented through modifications to the Spark scheduler and integrating HTCondor to manage opportunistic nodes. The architecture ensures minimal disruption to the standard Spark execution model while enabling dynamic resource elasticity and improved system robustness.

The system was evaluated using Intel HiBench benchmarks and a real-world use case in genome analysis, demonstrating significant improvements in resource efficiency and execution time under hybrid cluster configurations. Notably, the OPERA-gSAM deployment extended BigOPERA by integrating Google Cloud Storage (GCS) support, showcasing the adaptability of the framework to different storage backends and application domains.

The contributions of this work can be summarized as follows:

- The design and implementation of a hybrid scheduling architecture for Apache Spark, capable of distinguishing and leveraging dedicated and opportunistic resources.

- A novel HTCondor integration with Spark to dynamically manage non-dedicated nodes using Docker containers.

- Empirical validation of the proposed system through synthetic benchmarks, demonstrating performance, scalability, and resource utilization improvements under diverse workload conditions.

- Assessment of the suitability of the framework for real-world applications through a genomics case study, confirming its effectiveness in supporting data-intensive bioinformatics workflows.

Together, these contributions establish BigOPERA as a viable strategy for improving the elasticity and efficiency of big data frameworks through opportunistic computing, paving the way for more sustainable and cost-effective data processing infrastructures. By enabling the integration of non-dedicated resources into the execution pool, BigOPERA not only enhances resource utilization but also increases the overall reliability of the platform. As more opportunistic nodes become available, the system gains additional capacity to handle tasks, mitigating the impact of node failures or temporary unavailability. This increased redundancy contributes to greater fault tolerance and system robustness, especially under fluctuating workloads. Furthermore, the contribution to reducing the carbon footprint of data centres by utilizing otherwise idle resources also aligns with sustainable computing practices, highlighting the environmental relevance of this approach.

6.2 Recommendations for Future Research

Future work will build upon the current BigOPERA architecture to enhance its adaptability, energy efficiency, and broader applicability in Big Data processing environments. The following directions are proposed:

- **Enhanced Scheduling Logic:** Introduce adaptive logic into Spark internal scheduler to dynamically respond to runtime conditions such as node churn, task priority, and energy considerations. This would enable more flexible task-level decisions between dedicated and opportunistic resources.

- **Energy-Aware Resource Utilization:** Evaluate how opportunistic scheduling can incorporate energy efficiency metrics, identifying operating points that reduce power consumption without sacrificing throughput. This includes revisiting HTCondor configurations to activate resources based on energy-optimal thresholds rather than maximum CPU usage.
- **Energy Profiling for Scheduling:** Extend the resource model with energy efficiency metadata (e.g., performance-per-watt). This could inform more intelligent matchmaking and ranking strategies within HTCondor, encouraging the use of nodes that deliver high computational output at lower energy costs.
- **Predictive Resource Management:** Integrate machine learning techniques for predictive analytics on workload patterns and node availability, which could enhance the scheduler's ability to anticipate demand and proactively allocate resources.
- **Platform Generalization:** Expand BigOPERA to support other Big Data platforms beyond Spark, streaming, and real-time data processing frameworks.
- **Container Runtime Alternatives:** Explore the use of *Apptainer* (formerly Singularity) as a replacement for Docker to improve compatibility with HPC environments, enhance security in multi-user systems, and enable smoother integration in academic or restricted-access infrastructures.

These directions aim to consolidate BigOPERA as a flexible and sustainable resource management framework for Big Data, capable of adapting to a wide range of workloads and infrastructures while contributing to environmentally conscious computing.

APPENDIX A

PUBLICATIONS

- Articles directly related to this thesis:
  - Articles in peer-reviewed journals:
    - Pablo V. Caderno, Feras Awaysheh, José C. Cabaleiro, and Tomás F. Pena. "BigOPERA: An OPportunistic and Elastic Resource Allocation for big data frameworks". In: *Cluster Computing* (2025). Affiliations: Caderno, Pablo Vazquez (CiTIUS research center, University of Santiago de Compostela, Spain), Awaysheh, Feras M. (Department of Computing Science, ADSLab, Umeå University, Umeå, Sweden), José C. Cabaleiro (CiTIUS research center, University of Santiago de Compostela, Spain), Tomás F. Pena (CiTIUS research center, University of Santiago de Compostela, Spain). DOI: 10.1007/s10586-025-05274-4
      Impact factor (JCR 2023): 3.6, Q1.
      Category: Computer Science - Computer Networks and Communications.
      Rank: 50/395.
      Impact factor (SJR 2023): 1.069 Q1.
      Category: Computer Science - Software.
      Rank: 59/407.
      Contributions: Designed the two-tiered scheduling mechanism, and led the experimental design and data analysis. Contributed significantly to the manuscript's writing and final review.
  - Articles published in international conferences:

∗ Pablo Vazquez Caderno, Feras M. Awaysheh, Yolanda Colino-Sanguino, Laura Rodriguez De La Fuente, Fatima Valdes-Mora, Jose C. Cabaleiro, Tomas F. Pena, and David Gallego-Ortega. "OPERA-gSAM: Big Data Processing Framework for UMI Sequencing at High Scalability and Efficiency". In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*. Affiliations: Caderno, Pablo Vazquez (CiTIUS research center, University of Santiago de Compostela, Spain), Awaysheh, Feras M. (Institute of Computer Science, Delta Center University of Tartu, Tartu, Estonia), Colino-Sanguino, Yolanda (Cancer Epigenetic Biology and Therapeutics Laboratory, Children's Cancer Institute, Sydney, Australia School of Clinical Medicine, University of New South Wales, Sydney, Australia), Laura Rodriguez De La (Cancer Epigenetic Biology and Therapeutics Laboratory, Children's Cancer Institute, Sydney, Australia The Kinghorn Cancer Centre, Garvan Institute of Medical Research, Sydney, Australia St. Vincent's Clinical School, Faculty of Medicine, University of New South Wales, Sydney, Australia), Valdes-Mora, Fatima (Cancer Epigenetic Biology and Therapeutics Laboratory, Children's Cancer Institute, Sydney, Australia School of Clinical Medicine, University of New South Wales, Sydney, Australia), Cabaleiro, Jose C. (CiTIUS research center, University of Santiago de Compostela, Spain), Pena, Tomas F. (CiTIUS research center, University of Santiago de Compostela, Spain), Gallego-Ortega, David (The Kinghorn Cancer Centre, Garvan Institute of Medical Research, Sydney, Australia St. Vincent's Clinical School, Faculty of Medicine, University of New South Wales, Sydney, Australia SC of Biomedical Engineering, FAC of Engineering and Information Technology, University of Technology, Sydney, Australia). 2023, pp. 160–167. DOI: 10.1109/CCGridW59191.2023.00038

Contributions: Design, development, and optimization of the algorithms and code. Performing the experiments, analyzing the results, and writing the article.

GCS Class: 2.

GCS Rating: A.

A.1 Journal Authorizations

OPERA-gSAM: Big Data Processing Framework for UMI Sequencing at High Scalability and Efficiency

Conference Proceedings:
2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)

Author: Pablo Vazquez Caderno

Publisher: IEEE

Date: May 2023

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/ educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Licence to Publish - Open Access

SPRINGER NATURE

Licensee:	Springer Science+Business Media LLC	(the 'Licensee')
Journal Name:	Cluster Computing	(the 'Journal')
Manuscript Number:	1d67650d-0e05-4923-a4f0-1c39ba2baf9b	
Proposed Title of Article:	BigOPERA: An OPportunistic and Elastic Resource Allocation for big data frameworks	(the 'Article')
Author(s) [Please list all named Authors]:	Pablo V. Caderno, Feras Awaysheh, José C. Cabaleiro, Tomás F. Pena	(the 'Author')
Corresponding Author Name:	Feras Awaysheh	

Licence Applicable to the Article:

Creative Commons licence CC BY: This licence permits use, duplication, adaptation, distribution and reproduction in any medium or format, as long as appropriate credit is given to the original author(s) and the source, a link is provided to the Creative Commons licence, and any changes made are indicated. Please read the full licence for further details at - http://creativecommons.org/licenses/by/4.0/

Subject to editorial acceptance of the Article, it will be published under the Creative Commons licence shown above.

1 Grant of Rights

a) For good and valuable consideration, the Author hereby grants to the Licensee the perpetual, non-exclusive, irrevocable, world-wide, assignable, sublicensable and unlimited right to: publish, reproduce, copy, distribute, communicate, display publicly, sell, rent and/ or otherwise make available the article identified above, including any supplementary information and graphic elements therein (e.g. illustrations, charts, moving images) (the "Article") in any language, in any versions or editions in any and all forms and/or media of expression (including without limitation in connection with any and all end-user devices), whether now known or developed in the future. Without limitation, the above grant includes: (i) the right to edit, alter, adapt, adjust and prepare derivative works; (ii) all commercial use, advertising, and marketing rights, including without limitation graphic elements on the cover of the journal and in relation to social media; (iii) rights for any training, educational and/or instructional purposes; (iv) the right to add and/or remove links or combinations with other media/works; and (v) the right to create, use and/or license and/or sublicense content data or metadata of any kind in relation to the Article (including abstracts and summaries) without restriction. The above rights are granted in relation to the Article as a whole or any part and with or in relation to any other works.

b) Without limiting the rights granted above, Licensee is granted the rights to use the Article for the purposes of analysis, testing, and development of publishing- and research-related workflows, systems, products, projects, and services; to confidentially share the Article with select third parties to do the same; and to retain and store the Article and any associated correspondence/files/forms to maintain the historical record, and to facilitate research integrity investigations. The grant of rights set forth in this clause (b) is irrevocable.

c) The Licensee will have the right, but not the obligation, to exercise any or all of the rights granted herein. If the Licensee elects not to publish the Article for any reason, all publishing rights under this Agreement as set forth in clause 1.a) above will revert to the Author.

2 Copyright

Ownership of copyright in the Article will be vested in the name of the Author. When reproducing the Article or extracts from it, the Author will acknowledge and reference first publication in the Journal.

3 Use of Article Versions

a) For purposes of this Agreement: (i) references to the "Article" include all versions of the Article; (ii) "Submitted Manuscript" means the version of the Article as first submitted by the Author; (iii) "Accepted Manuscript" means the version of the Article accepted for publication, but prior to copy-editing and typesetting; and (iv) "Version of Record" means the version of the Article published by the Licensee, after copy-editing and typesetting. Rights to all versions of the Manuscript are granted on a non-exclusive basis.

b) The Author may make the Submitted Manuscript available at any time and under any terms (including, but not limited to, under a CC BY licence), at the Author's discretion. Once the Article has been published, the Author will include an acknowledgement and provide a link to the Version of Record on the publisher's website: "This preprint has not undergone peer review (when applicable) or any post-submission improvements or corrections. The Version of Record of this article is published in [insert journal title], and is available online at https://doi.org/[insert DOI]".

c) Immediately after acceptance the Author may deposit the Accepted Manuscript to any location, and under any terms (including, but not limited to, under a CC BY licence), provided it is not made publicly available until after publication. The Author will include an acknowledgement in the Accepted Manuscript, together with a link to the Version of Record on the publisher's website: "This version of the article has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/[insert DOI]".

4 Warranties & Representations

Author warrants and represents that:

a)

i. the Author is the sole copyright owner or has been authorised by any additional copyright owner(s) to grant the rights defined in clause 1,

ii. the Article does not infringe any intellectual property rights (including without limitation copyright, database rights or trade mark rights) or other third party rights and no licence from or payments to a third party are required to publish the Article,

iii. the Article has not been previously published, nor has the Author committed to licensing any version of the Article under a licence inconsistent with the terms of this Agreement,

iv. if the Article contains materials from other sources (e.g. illustrations, tables, text quotations), Author has obtained written permissions to the extent necessary from the copyright holder(s), to license to the Licensee the same rights as set out in clause 1 and has cited any such materials correctly;

b) all of the facts contained in the Article are according to the current body of research true and accurate;

c) nothing in the Article is obscene, defamatory, violates any right of privacy or publicity, infringes any other human, personal or other rights of any person or entity or is otherwise unlawful and that informed consent to publish has been obtained for any research participants;

d) nothing in the Article infringes any duty of confidentiality owed to any third party or violates any contract, express or implied, of the Author;

e) all institutional, governmental, and/or other approvals which may be required in connection with the research reflected in the Article have been obtained and continue in effect;

f) all statements and declarations made by the Author in connection with the Article are true and correct; and

g) the signatory who has signed this agreement has full right, power and authority to enter into this agreement on behalf of all of the Authors.

5 Cooperation

a) The Author will cooperate fully with the Licensee in relation to any legal action that might arise from the publication of the Article, and the Author will give the Licensee access at reasonable times to any relevant accounts, documents and records within the power or control of the Author. The Author agrees that any Licensee affiliate through which the Licensee exercises any rights or performs any obligations under this Agreement is intended to have the benefit of and will have the right to enforce the terms of this Agreement.

b) Author authorises the Licensee to take such steps as it considers necessary at its own expense in the Author's name(s) and on their behalf if the Licensee believes that a third party is infringing or is likely to infringe copyright in the Article including but not limited to initiating legal proceedings.

6 Author List

Changes of authorship, including, but not limited to, changes in the corresponding author or the sequence of authors, are not permitted after acceptance of a manuscript.

7 Post Publication Actions

The Author agrees that the Licensee may remove or retract the Article or publish a correction or other notice in relation to the Article if the Licensee determines that such actions are appropriate from an editorial, research integrity, or legal perspective.

8 Controlling Terms

The terms of this Agreement will supersede any other inconsistent terms that the Author or any third party may assert apply to any version of the Article.

9 Governing Law

This Agreement will be governed by, and construed in accordance with, the laws of New York State. The courts of New York, N.Y. will have exclusive jurisdiction.

Signed for and on behalf of the Author(s)
Corresponding Author: Feras Awaysheh
Email: feras.awaysheh@umu.se
IP Address: 130.239.36.226

Time Stamp: 2025-05-09 09:09:37

Bibliography

[1] S. Suneetha and M. Usha Rani. "Swift Towards Big Data Analytics". In: *International journal of engineering research and technology* 2 (2018). URL: https://api.semanticscholar.org/CorpusID:86773783.

[2] Nand Kumar et al. "Harnessing the Power of Big Data: Challenges and Opportunities in Analytics". In: *Tuijin Jishu/Journal of Propulsion Technology* 44 (Sept. 2023), pp. 1001–4055.

[3] Ramatu Nda and Rosmaini Tasmin. "Big Data Management in Education Sector: an Overview". In: *Path of Science* 5.6 (2019), pp. 5009–5014. ISSN: 2413-9009. DOI: 10.22178/pos.47-6. URL: https://pathofscience.org/index.php/ps/article/view/620.

[4] Bran Knowles. *ACM TechBrief: Computing and Climate Change*. New York, NY, USA, 2021.

[5] University of Glasgow. *Computing Emissions Could Exceed Carbon Budget by 2040*. https://www.gla.ac.uk/news/archiveofnews/2024/september/headline_1111841_en.html. 2024.

[6] Douglas Thain, Todd Tannenbaum, and Miron Livny. "Distributed computing in practice: the Condor experience". In: *Concurrency and Computation: Practice and Experience* 17.2-4 (2005), pp. 323–356. DOI: 10.1002/cpe.938.

[7] Abderrahmen Mtibaa et al. "Towards Mobile Opportunistic Computing". In: *2015 IEEE 8th International Conference on Cloud Computing*. 2015, pp. 1111–1114. DOI: 10.1109/CLOUD.2015.163.

[8] J.W. Strickland et al. “Governor: Autonomic Throttling for Aggressive Idle Resource Scavenging”. In: *Second International Conference on Autonomic Computing (ICAC’05)*. 2005, pp. 64–75. DOI: 10.1109/ICAC.2005.31.

[9] *Apache Spark - Unified Engine for large-scale data analytics — spark.apache.org*. https://spark.apache.org/. [Accessed 17-06-2024].

[10] *CERN Batch Service User Guide*. https://batchdocs.web.cern.ch/. [Accessed 17-06-2024].

[11] David Bernstein. “Containers and Cloud: From LXC to Docker to Kubernetes”. In: *IEEE Cloud Computing* 1.3 (2014), pp. 81–84. DOI: 10.1109/MCC.2014.51.

[12] Brendan Burns et al. “Borg, Omega, and Kubernetes”. In: *Commun. ACM* 59.5 (Apr. 2016), pp. 50–57. ISSN: 0001-0782. DOI: 10.1145/2890784. URL: https://doi.org/10.1145/2890784.

[13] Amazon Australia. *The Climate Pledge*. Accessed September 17, 2024. Amazon Australia, 2024. URL: https://www.aboutamazon.com.au/planet/climate-pledge.

[14] Amazon Web Services. *Amazon Web Services: 2023 Sustainability Report Summary*. Sustainability Report. Accessed September 17, 2024. Amazon, 2023. URL: https://sustainability.aboutamazon.com/content/dam/sustainability-marketing-site/pdfs/reports-docs/2023-amazon-sustainability-report-aws-summary.pdf.

[15] United Nations Framework Convention on Climate Change. *Microsoft: Carbon Negative Goal*. Climate Neutral Now. Accessed September 17, 2024. UNFCCC, 2024. URL: https://unfccc.int/climate-action/un-global-climate-action-awards/climate-neutral-now/microsoft-carbon-negative-goal.

[16] Microsoft Corporation. *Annual Report 2023*. Annual Report. Accessed September 17, 2024. Microsoft, 2023. URL: https://www.microsoft.com/investor/reports/ar23/index.html.

[17] Google. *Our third decade of climate action: Realizing a carbon-free future*. Blog post. Accessed September 17, 2024. Google, 2020. URL: https://blog.google/outreach-initiatives/sustainability/our-third-decade-climate-action-realizing-carbon-free-future/.

[18] Fizza Iftikhar, Ayesha Asghar, and Maha Khan. “The Impact of Green Marketing and Environmental Awareness on Consumers Green and Conscious Consumption of Green Products”. In: *Academic Journal of Social Sciences (AJSS)* 6 (June 2022), pp. 077–094. DOI: 10.54692/ajss.2022.06021695.

[19] Renyu Yang et al. “Performance-Aware Speculative Resource Oversubscription for Large-Scale Clusters”. In: *IEEE Transactions on Parallel and Distributed Systems* 31.7 (2020), pp. 1499–1517. DOI: 10.1109/TPDS.2020.2970013.

[20] Francis X. Diebold. “On the Origin(s) and Development of the Term ’Big Data’”. In: *PIER Working Paper No. 12-037* (2012). URL: https://ssrn.com/abstract=2152421.

[21] Douglas Laney. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. Tech. rep. META Group, 2001. URL: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[22] Rob Kitchin and Gavin McArdle. “What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets”. In: *Big Data & Society* 3 (Feb. 2016). DOI: 10.1177/2053951716631130.

[23] Martin Dodge and Rob Kitchin. “Codes of life: Identification codes and the machine-readable world”. In: *Environment and Planning D: Society and Space* 23.6 (2005), pp. 851–881.

[24] Danah Boyd and Kate Crawford. “Critical Questions for Big Data: Provocations for a cultural, technological, and scholarly phenomenon”. In: *Information, Communication & Society* 15.5 (2012), pp. 662–679.

[25] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable real-time data systems*. Manning Publications, 2012.

[26] Viktor Mayer-Schönberger and Kenneth Cukier. *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. John Murray Publishers338 Euston Road-LondonUnited Kingdom, 2013.

[27] Bernard Marr. *Big data: The 5 vs everyone must know*. https://www.linkedin.com/pulse/20140306073407-64875646-big-data-the-5-vs-everyone-must-know. 2014.

[28] Eileen McNulty. *Understanding Big Data: The Seven V's*. https://dataconomy.com/2014/05/22/seven-vs-big-data/. 2014.

[29] Seth Grimes. *Big Data: Avoid 'Wanna V' Confusion*. Accessed on March 2025. 2013. URL: https://www.informationweek.com/machine-learning-ai/big-data-avoid-wanna-v-confusion.

[30] Dough Laney. *Beyond Volume, Variety and Velocity is the Issue of Big Data Veracity*. Accessed on March 2025. 2013. URL: https://insideainews.com/2013/09/12/beyond-volume-variety-velocity-issue-big-data-veracity/#comment-795.

[31] Wasnaa Kadhim Jawad and Abbas M. Al-Bakry. "Big Data Analytics: A Survey". In: *International Journal of Computers and Informatics* 49.1 (2023). Accessed: 2025-05-16. DOI: 10.25195/ijci.v49i1.384. URL: https://doi.org/10.25195/ijci.v49i1.384.

[32] A. McAfee et al. "Big data: The management revolution". In: *Harvard Bus Rev* 90 (Jan. 2012), pp. 61–67.

[33] Amir Gandomi and Murtaza Haider. "Beyond the hype: Big data concepts, methods, and analytics". In: *International Journal of Information Management* 35.2 (2015), pp. 137–144. ISSN: 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2014.10.007. URL: https://www.sciencedirect.com/science/article/pii/S0268401214001066.

[34] Kornelia Batko and Andrzej Ślęzak. "The use of Big Data Analytics in healthcare". In: *Journal of Big Data* 9.1 (Jan. 2022), p. 3. ISSN: 2196-1115. DOI: 10.1186/s40537-021-00553-4. URL: https://doi.org/10.1186/s40537-021-00553-4.

[35] Itay Goldstein, Chester S. Spatt, and Mao Ye. *Big Data in Finance*. SSRN Working Paper. Mar. 2021. DOI: 10.2139/ssrn.3809447. URL: https://ssrn.com/abstract=3809447.

[36] N.N. Misra et al. "IoT, Big Data, and Artificial Intelligence in Agriculture and Food Industry". In: *IEEE Internet of Things Journal* PP (May 2020), pp. 1–1. DOI: 10.1109/JIOT.2020.2998584.

[37] danah boyd and Kate Crawford and. “CRITICAL QUESTIONS FOR BIG DATA”. In: *Information, Communication & Society* 15.5 (2012), pp. 662–679. DOI: 10.1080/1369118X.2012.678878. eprint: https://doi.org/10.1080/1369118X.2012.678878. URL: https://doi.org/10.1080/1369118X.2012.678878.

[38] William Yu Chung Wang and Yichuan Wang. “Analytics in the era of big data: The digital transformations and value creation in industrial marketing”. In: *Industrial Marketing Management* 86 (2020), pp. 12–15. ISSN: 0019-8501. DOI: https://doi.org/10.1016/j.indmarman.2020.01.005. URL: https://www.sciencedirect.com/science/article/pii/S0019850120300298.

[39] Matei Zaharia et al. “Spark: cluster computing with working sets”. In: *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud’10. Boston, MA: USENIX Association, 2010, p. 10.

[40] Feras M Awaysheh and Tomás F Pena. “EME: An Automated, Elastic and Efficient Prototype for Provisioning Hadoop Clusters On-demand.” In: 2017.

[41] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: https://doi.org/10.1145/1327452.1327492.

[42] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google File System”. In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), pp. 29–43. DOI: 10.1145/945445.945450.

[43] The Apache Software Foundation. *Apache Hadoop*. Accessed April 17, 2025. 2025. URL: https://hadoop.apache.org/.

[44] Toshifa Hussain, Anirudh Sanga, and Shweta Mongia. “Big Data Hadoop Tools and Technologies: A Review”. In: Available at SSRN: https://ssrn.com/abstract=3462554. Proceedings of International Conference on Advancements in Computing & Management (ICACM) 2019, 2019. DOI: http://dx.doi.org/10.2139/ssrn.3462554.

[45] *Apache Hadoop — hadoop.apache.org*. https://hadoop.apache.org/release/2.2.0.html. [Accessed 01-05-2025]. 20013.

[46] " Co-authored by Bharadwaj Jayaraman and Manoj Kumar". *Scalable Automated Config-driven data Validation with ValiData — linkedin.com*. https://www.linkedin.com/blog/engineering/data-management/scalable-automated-config-driven-data-validation. [Accessed 01-05-2025].

[47] Abhi Khune et al. *Modernizing Uber's Batch Data Infrastructure with Google Cloud Platform*. https://www.uber.com/en-AU/blog/modernizing-ubers-data-infrastructure-with-gcp/. [Accessed 01-05-2025].

[48] Matei Zaharia et al. "Spark: Cluster computing with working sets." In: *HotCloud* 10.10-10 (2010), p. 95.

[49] Matt W. Mutka and Miron Livny. "Profiling Workstations' Available Capacity for Remote Execution". In: *Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*. Performance '87. NLD: North-Holland Publishing Co., 1987, pp. 529–544. ISBN: 0444703470.

[50] D. Nichols. "Using idle workstations in a shared computing environment". In: *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*. SOSP '87. Austin, Texas, USA: Association for Computing Machinery, 1987, pp. 5–12. ISBN: 089791242X. DOI: 10.1145/41457.37502. URL: https://doi.org/10.1145/41457.37502.

[51] Matt Walter Mutka. *Sharing in a Privately Owned Workstation Environment*. Technical Report 781. Computer Sciences Technical Report #781. University of Wisconsin-Madison: Computer Sciences Department, July 1988.

[52] Douglas Thain, Todd Tannenbaum, and Miron Livny. "Distributed computing in practice: the Condor experience". In: *Concurrency and Computation: Practice and Experience* 17.2-4 (2005), pp. 323–356. DOI: https://doi.org/10.1002/cpe.938. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.938. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.938.

[53] HTCondor Team. *HTCondor Documentation*. https://htcondor.readthedocs.io/en/latest/. Accessed: [Insert Access Date Here]. 2023. URL: https://htcondor.readthedocs.io/en/latest/.

[54] Ruth Pordes et al. "The open science grid". In: *Journal of Physics: Conference Series* 78 (Aug. 2007), p. 012057. DOI: 10.1088/1742-6596/78/1/012057.

[55] R. Raman, M. Livny, and M. Solomon. "Matchmaking: distributed resource management for high throughput computing". In: *Proceedings. The Seventh International Symposium on High Performance Distributed Computing (Cat. No.98TB100244)*. 1998, pp. 140–146. DOI: 10.1109/HPDC.1998.709966.

[56] "A Comprehensive Analysis of Process Energy Consumption on Multi-Socket Systems with GPUs". In.

[57] Jordi Arjona et al. "A Measurement-based Analysis of the Energy Consumption of Data Center Servers". In: *Proceedings of the 5th International Conference on Future Energy Systems*. ACM, 2014, pp. 63–74. DOI: 10.1145/2602044.2602061.

[58] D Aruna and G Padmavathi. "Green Computing: Need of the Hour". In: *International Journal of Computer Trends and Technology (IJCTT)* 14.2 (2014), pp. 50–54.

[59] Arman Shehabi, Sarah Smith, and Eric Masanet. *United States Data Center Energy Usage Report*. Tech. rep. LBNL-2001556. Retrieved from user-uploaded material. Lawrence Berkeley National Laboratory, 2024.

[60] International Energy Agency. *Global Energy Review 2025*. https://www.iea.org/reports/global-energy-review-2025. 2025.

[61] Wu-chun Feng. *The Green Computing Book*. CRC Press, 2018. ISBN: 9781439819876.

[62] The White House. *The Trump Effect: A Running List of New U.S. Investment in President Trump's Second Term*. https://www.whitehouse.gov/articles/2025/05/trump-effect-a-running-list-of-new-u-s-investment-in-president-trumps-second-term/. Accessed: May 7, 2025. 2025.

[63] OpenAI and SoftBank. *Announcing the Stargate Project*. https://openai.com/index/announcing-the-stargate-project/. Accessed: May 7, 2025. 2025.

[64] JB Baker. *The Environmental Impact of OpenAI's Stargate Project*. https://builtin.com/articles/stargate-project-environmental-impact. Accessed: May 7, 2025. 2025.

[65] Feras M Awaysheh et al. "Security by design for big data frameworks over cloud computing". In: *IEEE Transactions on Engineering Management* 69.6 (2021), pp. 3676–3693.

[66] Apache Software Foundation. *Apache Hadoop*. https://hadoop.apache.org. [Accessed 10/22/2024].

[67] Heshan Lin et al. "MOON: MapReduce On Opportunistic ENvironments". In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC '10. Chicago, Illinois: Association for Computing Machinery, 2010, pp. 95–106. ISBN: 9781605589428. DOI: 10.1145/1851476.1851489. URL: https://doi.org/10.1145/1851476.1851489.

[68] Yi Yao et al. "OpERA: Opportunistic and Efficient Resource Allocation in Hadoop YARN by Harnessing Idle Resources". In: *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. 2016, pp. 1–9. DOI: 10.1109/ICCCN.2016.7568553.

[69] Matei Zaharia et al. "Improving MapReduce Performance in Heterogeneous Environments". In: *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*. OSDI'08. San Diego, California: USENIX Association, 2008, pp. 29–42.

[70] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[71] Ivanilton Polato et al. "A comprehensive view of Haddop research-A systematic literature review". In: *Journal of Network and Computer Applications* 46 (2014), pp. 1–25. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2014.07.022.

[72] Kamalakant Bawankule, Anil Kumar Singh, and Rupesh Kumar Dewaang. "Analysis of Task Scheduling in Hadoop MapReduce Framework". In: *Australian Journal of Wireless Technologies, Mobility and Security* 1 (Dec. 2022). URL: https://ausjournal.com/index.php/j/article/view/19.

[73] Mohamed Ragab, Feras M Awaysheh, and Riccardo Tommasini. "Bench-ranking: a first step towards prescriptive performance analyses for big data frameworks". In: *2021 IEEE international conference on big data (Big Data)*. IEEE. 2021, pp. 241–251.

[74] Tahseen Khan et al. "Machine learning (ML)-centric resource management in cloud computing: A review and future directions". In: *Journal of Network and Computer Applications* 204 (2022), p. 103405.

[75] Tahseen Khan et al. “Workload forecasting and energy state estimation in cloud data centres: ML-centric approach”. In: *Future Generation Computer Systems* 128 (2022), pp. 320–332.

[76] Benjamin Hindman et al. “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center”. In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI’11. Boston, MA: USENIX Association, 2011, pp. 295–308.

[77] Kay Ousterhout et al. “Sparrow: Distributed, Low Latency Scheduling”. In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP’13. Farminton, Pennsylvania: Association for Computing Machinery, 2013, pp. 69–84. ISBN: 9781450323888. DOI: 10.1145/2517349.2522716.

[78] Amandeep Verma and Sakshi Kaushal. “Cost-Time Efficient Scheduling Plan for Executing Workflows in the Cloud”. In: *Journal of Grid Computing* 13 (2015), pp. 495–506. DOI: 10.1007/s10723-015-9344-9.

[79] Ying Yan et al. “TR-Spark: Transient Computing for Big Data Analytics”. In: *Proceedings of the Seventh ACM Symposium on Cloud Computing*. SoCC ’16. Santa Clara, CA, USA: Association for Computing Machinery, 2016, pp. 484–496. ISBN: 9781450345255. DOI: 10.1145/2987550.2987576.

[80] Udit Mehrotra. *Spark enhancements for elasticity and resiliency on Amazon EMR*. https://aws.amazon.com/blogs/big-data/spark-enhancements-for-elasticity-and-resiliency-on-amazon-emr/. [Accessed 10/22/2024]. 2019.

[81] Prakash Chockalingam et al. *Introducing Databricks Optimized Autoscaling on Apache Spark™*. https://www.databricks.com/blog/2018/05/02/introducing-databricks-optimized-auto-scaling.html. [Accessed 10/22/2024]. 2018.

[82] Apache Software Foundation. *Hadoop: Fair Scheduler*. https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/FairScheduler.html. [Accessed 10/22/2024].

[83] *HTCondor — research.cs.wisc.edu*. https://research.cs.wisc.edu/htcondor/news/HTCondor_8.3.6_released!/. [Accessed 29-05-2025].

[84] *Overview - Spark 3.5.4 Documentation — spark.apache.org*. https://spark.apache.org/docs/3.5.4/index.html.

[85] *Tuning - Spark 3.2.1 Documentation — archive.apache.org*. https://archive.apache.org/dist/spark/docs/3.2.1/tuning.html#level-of-parallelism.

[86] Shengsheng Huang et al. "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis". In: *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. 2010, pp. 41–51. DOI: 10.1109/ICDEW.2010.5452747.

[87] Google Cloud. *General-purpose machine family for Compute Engine. Compute Engine Documentation.* https://cloud.google.com/compute/docs/general-purpose-machines#sharedcore. [Accessed 10/22/2024].

[88] Sachin Agarwal. *Google Cloud's E2 VMs compared to the N2 VMs*. https://www.bigbitbus.com/2021/06/10/Google-Cloud-E2-N2-VMs/. [Accessed 10/22/2024]. 2021.

[89] Bowen Yu et al. "Sparker: Efficient Reduction for More Scalable Machine Learning with Spark". In: *Proceedings of the 50th International Conference on Parallel Processing*. ICPP '21. Lemont, IL, USA: Association for Computing Machinery, 2021. ISBN: 9781450390682. DOI: 10.1145/3472456.3472499.

[90] Pablo V. Caderno et al. "BigOPERA: An OPportunistic and Elastic Resource Allocation for big data frameworks". In: *Cluster Computing* (2025). Affiliations: Caderno, Pablo Vazquez (CiTIUS research center, University of Santiago de Compostela, Spain), Awaysheh, Feras M. (Department of Computing Science, ADSLab, Umeå University, Umeå, Sweden), José C. Cabaleiro (CiTIUS research center, University of Santiago de Compostela, Spain), Tomás F. Pena (CiTIUS research center, University of Santiago de Compostela, Spain). DOI: 10.1007/s10586-025-05274-4.

[91] Pablo Vazquez Caderno et al. "OPERA-gSAM: Big Data Processing Framework for UMI Sequencing at High Scalability and Efficiency". In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*. Affiliations: Caderno, Pablo Vazquez (CiTIUS research center, University of Santiago de Compostela, Spain), Awaysheh, Feras M. (Institute of Computer Science, Delta Center University of Tartu, Tartu, Estonia), Colino-Sanguino, Yolanda

(Cancer Epigenetic Biology and Therapeutics Laboratory, Children's Cancer Institute, Sydney, Australia School of Clinical Medicine, University of New South Wales, Sydney, Australia), Laura Rodriguez De La (Cancer Epigenetic Biology and Therapeutics Laboratory, Children's Cancer Institute, Sydney, Australia The Kinghorn Cancer Centre, Garvan Institute of Medical Research, Sydney, Australia St. Vincent's Clinical School, Faculty of Medicine, University of New South Wales, Sydney, Australia), Valdes-Mora, Fatima (Cancer Epigenetic Biology and Therapeutics Laboratory, Children's Cancer Institute, Sydney, Australia School of Clinical Medicine, University of New South Wales, Sydney, Australia), Cabaleiro, Jose C. (CiTIUS research center, University of Santiago de Compostela, Spain), Pena, Tomas F. (CiTIUS research center, University of Santiago de Compostela, Spain), Gallego-Ortega, David (The Kinghorn Cancer Centre, Garvan Institute of Medical Research, Sydney, Australia St. Vincent's Clinical School, Faculty of Medicine, University of New South Wales, Sydney, Australia SC of Biomedical Engineering, FAC of Engineering and Information Technology, University of Technology, Sydney, Australia). 2023, pp. 160–167. DOI: 10.1109/CCGridW59191.2023.00038.

[92] Teemu Kivioja et al. "Counting absolute numbers of molecules using unique molecular identifiers". en. In: *Nat. Methods* 9.1 (Nov. 2011), pp. 72–74.

[93] Tom Smith, Andreas Heger, and Ian Sudbery. "UMI-tools: modeling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy". In: *Genome research* 27.3 (2017), pp. 491–499.

[94] Alexander Dobin et al. "STAR: ultrafast universal RNA-seq aligner". In: *Bioinformatics* 29.1 (2013), pp. 15–21.

[95] Feras M Awaysheh et al. "Big data resource management & networks: Taxonomy, survey, and future directions". In: *IEEE Communications Surveys & Tutorials* (2021).

[96] Tom Smith, Andreas Heger, and Ian Sudbery. "UMI-tools: modeling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy". en. In: *Genome Res.* 27.3 (Mar. 2017), pp. 491–499.

[97] Petr Danecek et al. "Twelve years of SAMtools and BCFtools". In: *GigaScience* 10.2 (Feb. 2021). giab008. ISSN: 2047-217X. DOI: 10.1093/gigascience/giab008. eprint: https://academic.oup.com/gigascience/article-pdf/

10/2/giab008/36332246/giab008.pdf. URL: https://doi.org/10.1093/gigascience/giab008.

[98] Saiful Islam et al. “Quantitative single-cell RNA-seq with unique molecular identifiers”. en. In: *Nat. Methods* 11.2 (Feb. 2014), pp. 163–166.

[99] Evan Z Macosko et al. “Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets”. en. In: *Cell* 161.5 (May 2015), pp. 1202–1214.

[100] Qiye He, Jeff Johnston, and Julia Zeitlinger. “ChIP-nexus enables improved detection of in vivo transcription factor binding footprints”. en. In: *Nat. Biotechnol.* 33.4 (Apr. 2015), pp. 395–401.

[101] Kasper Karlsson et al. “Amplification-free sequencing of cell-free DNA for prenatal non-invasive diagnosis of chromosomal aberrations”. en. In: *Genomics* 105.3 (Mar. 2015), pp. 150–158.

[102] Tao Zhu et al. “ATAC-seq with unique molecular identifiers improves quantification and footprinting”. en. In: *Commun. Biol.* 3.1 (Nov. 2020), p. 675.

[103] Patrik L Ståhl et al. “Visualization and analysis of gene expression in tissue sections by spatial transcriptomics”. en. In: *Science* 353.6294 (July 2016), pp. 78–82.

[104] Christopher Vollmers et al. “Genetic measurement of memory B-cell recall using antibody repertoire sequencing”. en. In: *Proc. Natl. Acad. Sci. U. S. A.* 110.33 (Aug. 2013), pp. 13463–13468.

[105] Andrian Yang, Michael Troup, and Joshua WK Ho. “Scalability and validation of big data bioinformatics software”. In: *Computational and structural biotechnology journal* 15 (2017), pp. 379–386.

[106] Pamela H Russell et al. “A large-scale analysis of bioinformatics code on GitHub”. In: *PloS one* 13.10 (2018), e0205898.

[107] Kalyan Nagaraj, GS Sharvani, and Amulyashree Sridhar. “Emerging trend of big data analytics in bioinformatics: a literature review”. In: *International Journal of Bioinformatics Research and Applications* 14.1-2 (2018), pp. 144–205.

[108] Beth K Martin et al. “Optimized single-nucleus transcriptional profiling by combinatorial indexing”. In: *Nature protocols* 18.1 (2023), pp. 188–207.

[109] Mingxuan Gao et al. “Comparison of high-throughput single-cell RNA sequencing data processing pipelines”. In: *Briefings in Bioinformatics* 22.3 (2021), bbaa116.

[110] José M Abuín, Tomás F Pena, and Juan C Pichel. “PASTASpark: multiple sequence alignment meets Big Data”. In: *Bioinformatics* 33.18 (2017), pp. 2948–2950.

[111] José M Abuín et al. “BigBWA: approaching the Burrows–Wheeler aligner to Big Data technologies”. In: *Bioinformatics* 31.24 (2015), pp. 4003–4005.

[112] Xueqi Li et al. “High-performance genomic analysis framework with in-memory computing”. In: *ACM SIGPLAN Notices* 53.1 (2018), pp. 317–328.

[113] Matti Niemenmaa et al. “Hadoop-BAM: directly manipulating next generation sequencing data in the cloud”. In: *Bioinformatics* 28.6 (2012), pp. 876–877.

[114] José M Abuín et al. “Big data in metagenomics: Apache spark vs MPI”. In: *Plos one* 15.10 (2020), e0239741.

[115] Rosa Filgueira et al. “SparkFlow: Towards high-performance data analytics for spark-based genome analysis”. In: *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2022, pp. 1007–1016.

[116] Yu Liu et al. “scSpark XMBD: High-Performance scRNA-seq Data Processing with Spark”. In: *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE. 2021, pp. 1956–1962.

[117] Heng Li et al. “The Sequence Alignment/Map format and SAMtools”. en. In: *Bioinformatics* 25.16 (Aug. 2009), pp. 2078–2079.

[118] Laura Rodriguez de la Fuente et al. “Tumor dissociation of highly viable cell suspensions for single-cell omic analyses in mouse models of breast cancer”. en. In: *STAR Protoc.* 2.4 (Dec. 2021), p. 100841.

[119] Mohamed Ragab, Feras M. Awaysheh, and Riccardo Tommasini. “Bench-Ranking: A First Step Towards Prescriptive Performance Analyses For Big Data Frameworks”. In: *2021 IEEE International Conference on Big Data (Big Data)*. 2021, pp. 241–251. DOI: 10.1109/BigData52589.2021.9671277.

List of Figures

List of Tables

Acronyms and Abbreviations

BD Big Data 13, 24, 34–36, 38, 39

GCP Google Cloud Platform 55

GCS Google Cloud Storage 63

HPC High-Performance Computing vii, 19

NGS Next-Generation Sequencing 14, 63

PCR Polymerase Chain Reaction 63

RDDs resilient distributed datasets 28

scRNA-seq single-cell RNA sequencing 14, 70

STAR Spliced Transcripts Alignment to a Reference 64

UMIs Unique molecular identifiers 64

The increasing demand for data-intensive applications calls for more scalable, flexible, and sustainable resource management in Big Data frameworks. This thesis introduces BigOPERA, a hybrid resource allocation system integrating opportunistic computing into Apache Spark. By combining dedicated and non-dedicated resources, BigOPERA enhances elasticity and fault tolerance without requiring changes to Spark core or user applications.

The architecture leverages containerized execution and a two-tier scheduler to allocate resources dynamically based on availability. Experimental results confirm improvements in performance, efficiency, and environmental impact. BigOPERA offers a practical and sustainable extension to existing Spark deployments, paving the way for more adaptive and energy-aware cluster computing.