

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

---

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

## **Analisi e sviluppo di middleware DDS per la gestione dei consumi in sistemi HPC**

**Relatore**

Prof. Andrea Bartolini

**Candidato**

Giacomo Madella

Ottobre 2023



## **Abstract**

# Indice

<b>1</b>	<b>Introduzione</b>	<b>4</b>
1.1	Contributi . . . . .	5
<b>2</b>	<b>Power management</b>	<b>6</b>
2.1	Stato dell'arte . . . . .	6
2.1.1	interfacce in-band . . . . .	7
2.1.2	interfacce out-of-band . . . . .	7
2.1.3	Interfacce di alto livello . . . . .	7
2.2	Modello di power stack . . . . .	8
2.2.1	Workflow engine . . . . .	9
2.2.2	Job schedulers . . . . .	9
2.2.3	Resource Manager . . . . .	10
2.2.4	System Manager . . . . .	10
2.2.5	Job Manager . . . . .	11
2.2.6	Node Manager . . . . .	11
2.2.7	Monitor . . . . .	11
<b>3</b>	<b>REGALE</b>	<b>12</b>
3.1	Obbiettivi . . . . .	12
3.2	Power Stack . . . . .	12
3.3	Problematica . . . . .	13
<b>4</b>	<b>DDS &amp; RTPS</b>	<b>14</b>
4.1	Implementazione usasta . . . . .	14
4.2	DDS . . . . .	14
4.3	RTPS . . . . .	15
<b>5</b>	<b>Casi di studio e valutazioni</b>	<b>17</b>
5.1	Test . . . . .	17
5.1.1	Struttura dei test . . . . .	18
5.1.2	Impatto del numero di sub in un dominio . . . . .	21

5.1.3	Test-1 . . . . .	22
5.1.4	Test-2 . . . . .	23
5.1.5	Test-3 . . . . .	25
5.2	Risultati . . . . .	25
5.3	Modello . . . . .	25
5.4	Scheletro componenti . . . . .	25
<b>6</b>	<b>Conclusioni</b>	<b>27</b>

# Introduzione

Nel panorama scientifico e industriale contemporaneo, assistiamo ad una sempre più crescente domanda di capacità computazionali, alimentata anche dalla necessità di gestire, monitorare e analizzare ingenti quantità di dati. Infatti ambiti come simulazioni complesse, meteorologia, calcolo in tempo reale e le nuove realtà di intelligenza artificiale sono rese possibili solo grazie a tecnologie il cui unico scopo è quello di risolvere problemi avanzati in tempi brevi. In particolare si parla di High Performance Computing (HPC), o calcolo ad elevate prestazioni, di quelle tecnologie che utilizzano cluster di processori e componenti hardware ad alte prestazioni in grado di processare dati multi-dimensionali in modo simultaneo. I sistemi di *High Performance Computing*, per poter sopperire a queste richieste, necessitano di diversi nodi di calcolo, ognuno dei quali è composto da molteplici processori come CPU, GPU o TPU e memorie ad alte prestazioni. Questi nodi solitamente sono collegati tra loro da reti ad alta velocità che permettono sia ai software di schedulazione di suddividere i workflow in diversi nodi, che ai diversi job di comunicare durante la loro esecuzione. Inoltre per poter funzionare al meglio questi cluster solitamente sono supportati da sistemi di raffreddamento in grado di gestire le notevoli quantità di calore che prodotte dalle attività di calcolo.

La gestione energetica di sistemi HPC è diventata per questo motivo, una delle principali preoccupazioni, non solo a causa dei costi monetari, ma anche per la sostenibilità ambientale e per la progettazione di nuove generazioni[1] di supercomputer. Infatti perpendicolarmente all'aumento della potenza computazionale richiesta, le tecnologie associate allo sviluppo dei componenti primari dei processori, i componenti più energivori del sistema, si sono avvicinati ai loro limiti fisici. Questi ultimi hanno portato a delle difficoltà sempre più grandi nel ridimensionamento dei transistor, conseguito nella progressiva fine delle leggi di Denard e Moore[1]. Tali leggi, che avevano guidato l'industria informatica per decenni, prevedevano un consumo energetico costante al crescere della velocità e capacità computazionale. Quando questi sono venuti a mancare, il mantenimento e ancora di più lo sviluppo di nuove generazioni di sistemi è diventato un compito tutt'altro che banale, e con questi si sono resi necessari sempre più software in grado di automatizzarne la gestione. Infatti utilizzare efficientemente l'energia disponibile e ottimizzare le prestazioni delle applicazioni sotto un limite di potenza è diventata

una sfida, che ha richiesto soluzioni specifiche. Il concetto di Power Management è nato sotto questo contesto, definendo un modello software che ha il compito di gestire tutto ciò che riguarda potenza di sistemi di HPC. Per farlo sono stati definiti diversi attori ognuno con un compito specifico, e cercando di standardizzare le interazioni che questi componenti dovevano avere.

Questo Power-Stack ha però la necessità di avere una visione globale per la gestione energetica, al fine di permettere dove necessario di definire degli obiettivi e limiti di prestazioni e consumi. Inoltre deve essere definita un'interfaccia standard per poter interagire con i controlli hardware e software di sistemi HPC di diversi produttori. Mentre sono state proposte diverse tecniche per colmare questo bisogno, la maggior parte di esse si è rivelata essere una soluzione per soddisfare singoli obiettivi di ottimizzazione o per un singolo sistema di HPC. Infatti molti dei prodotti attualmente disponibili svolgono compiti a granularità diverse e spesso in conflitto gli uni con gli altri. Peraltro non sono neanche mai state definite o modellizzate interfacce di comunicazione tra i vari software, lasciando ai gestori dei sistemi di HPC, l'onere di farlo. L'obiettivo finale sarebbe infatti quello di fornire un middleware documentato e facilmente integrabile, nei vari strumenti ad oggi presenti, per collegarli tra loro utilizzando un approccio distribuito e sfruttando il potenziale del Data Distribution Service (DDS) nonché quello di Real-Time Publish-Subscribe (RTPS).

## **1.1 Contributi**

I contributi di questa tesi sono stati lo studio e caratterizzazione di una specifica implementazione di DDS all'interno di sistemi HPC al fine di fornire una visione più completa della possibilità di integrare questo strumento come base delle comunicazioni di un middleware per i componenti del power management. Successivamente sono stati valutati dei modelli basati su questi risultati come modo d'uso. Infine sono stati creati per completare il quadro due di questi attori, mancanti nelle implementazioni attualmente prodotte, utilizzando la libreria REGALE.

# Power management

I sistemi di HPC sono composti da decine di centinaia di nodi computazionali interconnessi tra di loro da reti a bassa latenza e alta banda. Tutti questi nodi uniti insieme, grazie alla grande quantità di processori e agli acceleratori che hanno montato al loro interno offrono capacità computazionali che nei giorni nostri hanno raggiunto ordini del ExaFlops ( $10^{18}$  operazioni di Floating Point per secondo). Visto l'aumento della densità energetica e alle magnifiche prestazioni raggiunte, la potenza necessaria per alimentare questi sistemi sta superando i 20MWatt. Se poi si considera che la maggior parte della potenza fornita, viene convertita in calore, si deve prendere in considerazione anche i consumi necessari per tenere raffreddati questi sistemi. Infatti se non adeguati, le difficili condizioni in cui lavorano comportano grandi inefficienze in termini di energia, che si traducono anche in degradazioni di prestazioni computazionali. Considerando tutto, ai centri che ospitano queste macchine si devono fornire decine di MWatt di potenza per ogni exa-supercomputer che hanno installato. Ordini di grandezza di questo tipo non sono facilmente ottenibili e hanno costi estremamente elevati. Al fine di definire dei power budget, e utilizzare efficientemente la potenza richiesta si sono resi necessari strumenti situati su diversi livelli di astrazione. Sono nati così i primi concetti di Power Management, componenti per controllare l'utilizzo di energia utilizzando diverse strategie al fine di ridurre gli sprechi energetici e, allo stesso tempo, garantire un temperatura di funzionamento sicura. Power Management può essere visto come un sistema composto sia da parti software che Hardware. L'insieme di questi componenti va a formare un power-stack in grado di gestire la potenza assorbita di macchine HPC.

## 2.1 Stato dell'arte

Per riuscire nel suo scopo, il Power-Stack deve riuscire a gestire la potenza dei sistemi anche a granularità più basse, come singoli nodi, e singoli elementi all'interno dei nodi fornendo servizi a diversi livelli di astrazione. Tra questi vediamo:

- servizi in-band,



- servizi out-of-band
- applicazioni utente, e componenti di alto livello

### **2.1.1 interfacce in-band**

Le interfacce in-band sono rese possibili tramite componenti hardware sul chip, come i power knob (manopole per la gestione della potenza) che controlla il consumo e le prestazioni di ogni processore, e sensori che permettono di monitorare temperature e tensioni. Queste interfacce sono spesso utilizzate dai power governors (SO) e anche da alcune interfacce del System Management per permettere di limitare ad un massimo di potenza a livello di processore e a livello di nodo. Un esempio di questi componenti è il power controller che risiede all'interno dei processori, ed esponendo lo spazio di memoria interno a degli indirizzi di I/O riesce a fornire a chi lo necessita tutti i controlli. Grazie a questi componenti, sia il sistema operativo, che il System Manager riesce ad accedere a delle strutture dati contenenti gli stati, statistiche e consumi dei diversi componenti all'interno del chip.

### **2.1.2 interfacce out-of-band**

Sono interfacce che diversamente da quelle di prima, risiedono al di fuori dei singoli processori, ma comunque all'interno della scheda madre. Tra questi il più importante è il Board Management Controller, solitamente un microcontrollore out-of-band che esegue in modo indipendente dal processore. Il suo principale scopo è quello di monitorare in modo dettagliato lo stato di tensioni, temperature, ventole e prestazioni dei processori e fornire contemporaneamente servizi di power capping sia a livello di sistema che di singoli processori. Infine essendo alimentato in modo indipendente riesce a monitorare errori ed eccezioni che si manifestano sui processori, fornendo dettagli di elevata importanza.

### **2.1.3 Interfacce di alto livello**

Tra le principali interfacce di alto livello, a livello di sistema operativo troviamo i *power governors*, software in grado di gestire e controllare il consumo energetico dei componenti hardware sottostanti, ed in particolare quello dei processori. Svolgono principalmente due azioni: (i) monitoraggio del sistema in termini di carico di lavoro, temperatura e tensione e (ii) scelta della modalità di funzionamento in modo dinamico. In particolare questa ultima opzione prevede di adattarsi dinamicamente alle variazioni del carico di lavoro per fornire più o meno potenza ai processori.

Nel corso degli anni sono state sviluppati diversi software con compiti e meccanismi differenti. Tra questi rivediamo *Variorum* (LLNL), *GEOPM* (Intel)[2], and *HDEEM* (Atos)[3].

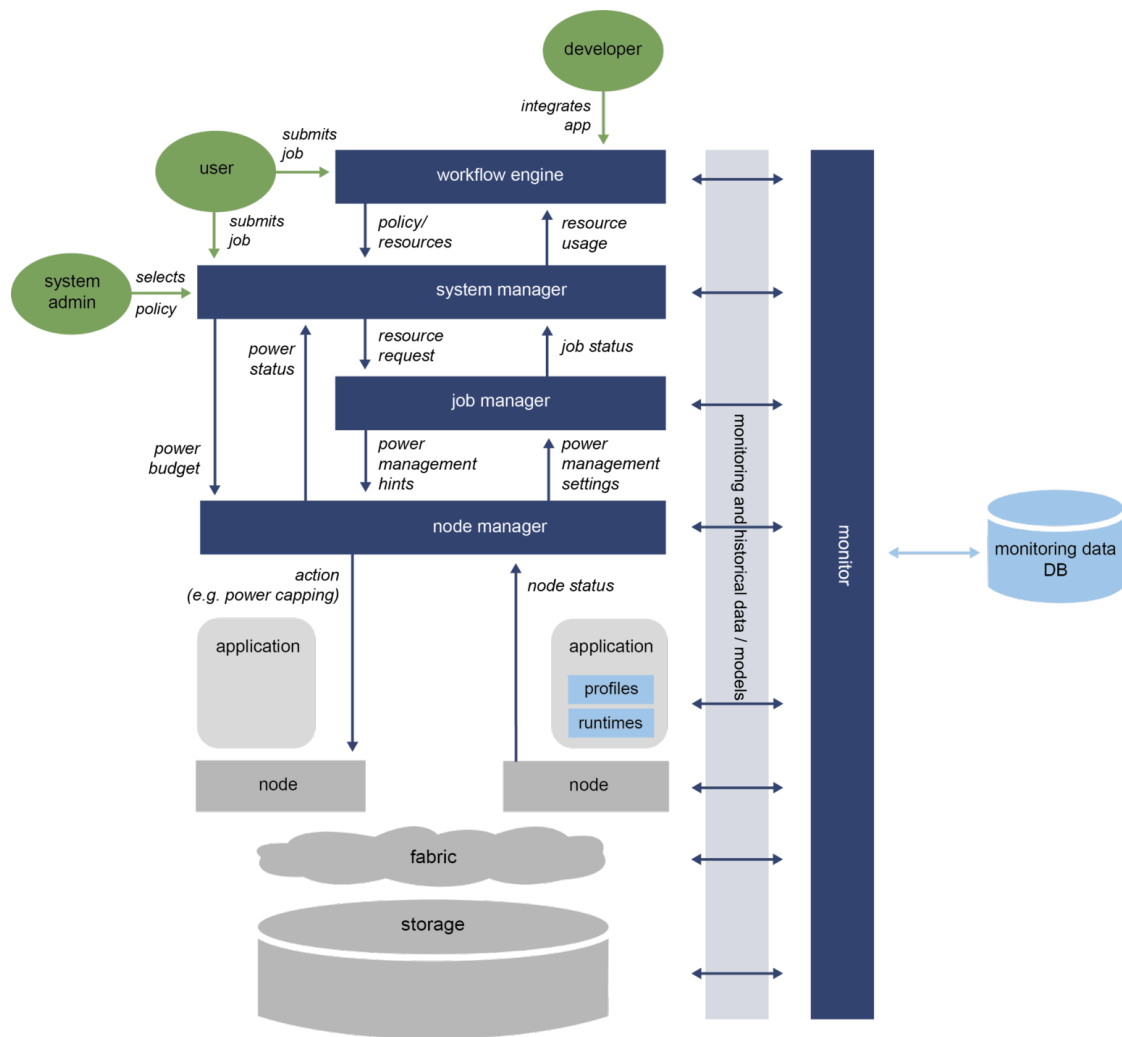
Tutti questi strumenti rappresentano un tentativo di risolvere un problema specifico di Power Management e non un software globale di gestione dell'energia di sistemi HPC.

## 2.2 Modello di power stack

Per utilizzare in modo controllato tutte le interfacce precedentemente spiegate, è stato necessario definire un modello di interazione e responsabilità dei vari componenti. Infatti un power stack completo e interoperabile, deve essere composto da attori che svolgono ruoli ben precisi e definiti. Tra questi i più importanti sono:

- Workflow engine
- System Manager
- Job Manager
- Node Manager
- Monitor

Di seguito viene riportato uno schema?? che mostra le interazioni tra i vari attori.



### 2.2.1 Workflow engine

Il workflow engine analizza le dipendenze e le richieste di risorse di ogni workflow e decide dinamicamente come dividerlo negli specifici jobs che verranno assegnati al system-manager.

### 2.2.2 Job schedulers

Il job scheduler ha il compito di assegnare e condividere le risorse computazionali e fisiche del sistema HPC, ai vari utenti che lo utilizzano. In particolare la serie di compiti che si trova a svolgere è il seguente L'utente schedula i jobs da svolgere in una o più code, definite dallo scheduler. Il Job scheduler esamina tutte le code e i job in esse con-

tenute, e decide dinamicamente, quale sarà l'ordine di esecuzione, e il tempo massimo in cui viene assegnata una risorsa. Generalmente si cerca di ottimizzare alcune caratteristiche come il tempo di utilizzo del sistema oppure l'accesso veloce alle risorse per alcuni sottoinsiemi di jobs. Inoltre le code definite, possono avere diverse priorità o può essere ristretto l'accesso a soli alcuni utenti. I job scheduler possono condividere un nodo anche con più utenti contemporaneamente, in base all'utilizzo che devono farne. Per farlo il nodo viene allocato e diviso in partizioni virtuali, che vengono sciolte una volta finiti i job in esecuzione. Questo permette di utilizzare al massimo i componenti messi a disposizione dal sistema HPC.

### 2.2.3 Resource Manager

Per riuscire a svolgere questo lavoro il Job Scheduler interagisce con uno o più **Resource Manager**. Questi sono software che hanno il privilegio di gestire le risorse di un centro di calcolo. Queste risorse includono diversi componenti:

- Nodi
- Processori
- Memorie
- Dischi
- Canali di comunicazione (compresi quelli di I/O)
- Interfacce di rete

Per esempio quando un Job Scheduler deve eseguire un job, richiede al RM di allocare core, memorie, dischi e risorse di rete in linea con quanto il job ha necessita di essere eseguito.

Infine in alcuni casi il RM è anche responsabile di gestire elettricità e raffreddamento dei centri di calcolo.

### 2.2.4 System Manager

Il System Manager è un componente che riceve come input un insieme di jobs che devono essere schedulati all'interno del sistema, e in modo indicativo decide quando schedulare ogni job, su quale nodo, e con quale power budget. Successivamente vengono monitorati i dati relativi a potenza ed energia, e controlla di conseguenza i budget di potenza, e la *user-fairness*

### **2.2.5 Job Manager**

Lo scopo del job manager è quello di effettuare ottimizzazioni job-centriche considerando le prestazioni di ogni applicazioni, il suo utilizzo di risorse, la sua fase e qualsiasi interazione dettata da ogni workflow in cui è presente. In breve il job manager decide i target delle manopole del Power Management, come (i) CPU power cap, (ii) CPU clock frequency oltre ad eseguire ottimizzazione del codice.

### **2.2.6 Node Manager**

Il node manager fornisce accesso ai controlli e monitoraggio hardware a livello del nodo. Volendo permette anche di definire delle policy di power management. Ha infine lo scopo di preservare integrità, sicurezza del nodo sia in termini informatici che fisici.

### **2.2.7 Monitor**

Il monitor è responsabile di collezionare tutte le metriche in-band e out-of-band che riguardando prestazioni, utilizzo e stato delle risorse, potenza ed energia. Tutto questo deve essere fatto con il minor impatto possibile sul sistema dove sta agendo, collezionando, aggregando e analizzando le metriche e dove necessario, scambiandole ad altri attori. A sua volta il *Monitor* è scomponibile in tre sotto-moduli:

- Gestione Firma che genera una firma che identifica univocamente il job;
- Estimatore che valuta le proprietà dei job o dello stato del sistema usando la firma generata precedentemente;
- Dashboard che fornisce le funzionalità da mostrare agli sviluppatori.

# REGALE

REGALE[4] è un progetto open source nato ad Aprile 2021 che ha come compito quello di costruire uno stack software scalabile per migliorare l'efficienza di sistemi Exascale<sup>1</sup> HPC. Questo progetto è particolarmente interessante nell'ambito del Power Management in quanto si è proposta di implementare la quasi totalità degli attori proposti nello stato dell'arte per quanto riguarda il Power Management di sistemi di HPC.

## 3.1 Obbiettivi

## 3.2 Power Stack

Molti dei componenti mostrati nel modello del Power Stack ?? sono stati sviluppati e resi disponibili nel progetto di REGALE. A questi mancano solo il *Workflow engine* ed un *Resource manager* specifico.

---

<sup>1</sup>Exascale: capace di eseguire operazioni nell'ordine di ExaFlops ( $10^{18}$ )

	Monitor				Node Manager	Job Manager	System Manager		Workflow Engine
	DB	Dashb oard	Estim ator	Sig Hndlr			RJMS	SPM	
SLURM							X		
OAR							X		
DCDB	X	X	/	/					
BEO	X	X			X			X	
BDPO						X			
EAR	X	X	?	?	X	X		X	
Melissa									X
RYAX									X
Examon	X	X	/	/					
COUNTDOWN						X			
PULPcontroller					X				
BeBiDa							X		
EPCM			/					/	

**Figura 3.1.** Copertura componenti REGALE

### 3.3 Problematica

Vista l'implementazione proposta, il problema con il quale si sta interfacciando il progetto può essere descritto con mancanza di interoperabilità. Ciò che manca in REGALE, è un layer di comunicazione che permetta ai vari attori di comunicare tra di loro. Infatti sono stati prima proposti diversi modelli di interazione tra i vari componenti a due a due, fino a quando si è deciso di provare a sviluppare un ulteriore middleware che standardizzasse le comunicazioni tra i vari attori, utilizzando un singolo strumento per tutti i componenti.

# DDS & RTPS

DDS (Data Distribution Service)[5] e RTPS (Real-Time Publish-Subscribe)[6] costituiscono due soluzioni fondamentali nel campo delle comunicazioni distribuite e real-time. Queste tecnologie svolgono un ruolo importante nella la trasmissione di dati tra dispositivi e applicazioni interconnesse, rivestendo particolare importanza in scenari complessi come i sistemi embedded, in IoT e applicazioni ad alte prestazioni come l'HPC (High-Performance Computing).

## 4.1 Implementazione usata

DDS e RTPS sono dei protocolli di comunicazione per specifici casi di utilizzo. Ci sono state diverse implementazioni di questi protocolli da diverse società e organizzazioni, come:

- FastDDS (eProsima)
- CycloneDDS (Oracle)
- ConnextDDS
- GurumDDS

e tante altre. In tutti i successivi capitoli verrà preso come riferimento FastDDS ed in particolare la sua versione 2.11.2 [7]. E' stato scelto di utilizzare questa implementazione dato il supporto per la comunicazione Real-Time, e le impostazioni delle Qualità del servizio(QoS) che la rendevano perfetta per un utilizzo su sistemi di HPC.

## 4.2 DDS

Data Distribution Service è un protocollo di comunicazione incentrato sullo scambio di dati per sistemi distribuiti. Questo si basa su modello chiamato Data-Centric Publish Subscribe (DCPS) I principali attori che vengono coinvolti sono:



- **Publisher:** responsabile della creazione e configurazione dei DataWriter. Il DataWriter è l'entità responsabile della pubblicazione effettiva dei messaggi. Ciascuno avrà un Topic assegnato sotto il quale vengono pubblicati i messaggi;
- **Subscriber:** responsabile di ricevere i dati pubblicati sotto i topic ai quali si iscrive. Serve uno o più oggetti DataReader, che sono responsabili di comunicare la disponibilità di nuovi dati all'applicazione;
- **Topic:** collega i DataWriter con i DataReader. È univoco all'interno di un dominio DDS;
- **Dominio:** utilizzato per collegare tutti i publisher e subscriber appartenenti a una o più domini di appartenenza, che scambiano dati sotto diversi topic. Il DomainParticipant funge da contenitore per altre entità DCPS, e svolge anche la funzione di costruttore di entità Publisher, Subscriber e Topic fornendo anche servizi di QoS;
- **Partizione:** costituisce un isolamento logico di entità all'interno dell'isolamento fisico offerto dal dominio;

Inoltre DDS definisce le cosiddette Qualità di Servizio (QoS policy) che servono configurare il comportamento di ognuno di questi attori.

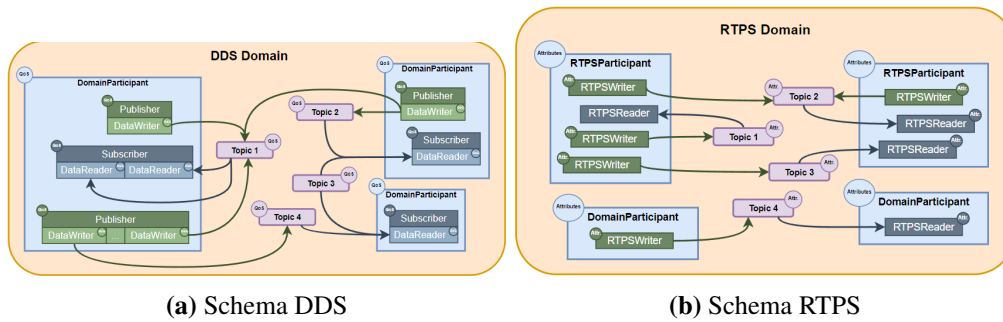
## 4.3 RTPS

Real-Time Publisher Subscribe protocol è un protocollo-middleware utilizzato da DDS per gestire la comunicazione su diversi protocolli di rete come UDP/TCP e Shared Memory. Il suo principale scopo è quello di inviare messaggi real-time, con un approccio best-effort e cercando di massimizzare l'efficienza. E' inoltre progettato per fornire strumenti per la comunicazione unicast e multicast. Le principali entità descritte da RTPS sono:

- **RTPSWriter:** endpoint capace di inviare dati;
- **RTPSReader:** endpoint abilitato alla ricezione dei dati;

Ereditato da DDS anche RTPS ha la concezione di Dominio di comunicazione e come questo, le comunicazioni a livello di RTPS girano attorno al concetto di Topic prima definito. L'unità di comunicazione è chiamata **Change** che rappresenta appunto un cambiamento sui dati scritti sotto un certo topic. Ognuno degli attori registra questi *Change* in una struttura dati che funge da cache. In particolare la sequenza di scambio è:

1. il *change* viene aggiunto nella cache del RTPSWriter;
2. RTPSWriter manda questa *change* a tutti gli RTPSReader che conosce;
3. quando RTPSReader riceve il messaggio, aggiorna la sua cache con il nuovo *change*.



**Figura 4.1.** Confronto tra architettura DDS e RTPS

# Casi di studio e valutazioni

In questa sezione verranno riportati i casi studio e i test effettuati sul framework. Tutti questi sono stati eseguiti su un sistema HPC Galielo-100 Cineca con le specifiche riportate nella tabella seguente

Parameter	Value
Number of nodes used	3
Processor	Intel CascadeLake 8260
Number of sockets per node	2
Number of cores per socket	24
Memory size per node	384 GB
Interconnect	Mellanox Infiniband 100GbE
OS	CentOS Linux
MPI	Open MPI 4.1.1

## 5.1 Test

Sono stati svolti diversi test al fine di trovare un modello ottimale di utilizzo e per la caratterizzazione di DDS, all'interno di sistemi HPC, nel contesto del Power Management. Nello specifico i test sono stati utili a capire il peso che avesse una singola configurazione o modello di utilizzo al fine di trovare quello più adeguato per una futura implementazione. I test effettuati sono:

- test-1: protocollo di comunicazione

- test-2: partizioni e wildcards
- test-3: throughput

Al fine di condurli nel modo più trasparente e corretto possibile sono stati resi pubblici [8] tutti i codici utilizzati durante lo svolgimento di questi test.

### **5.1.1 Struttura dei test**

Sono stati prodotti dei test ad hoc, composti in parte da script bash, e in parte da programmi c++ con lo scopo di sfruttare al massimo le infrastrutture utilizzate 5. Ognuno di questi, con il fine di coprire e reperire il maggior numero di informazioni possibili, ha dovuto monitorare e catalogare 4 tipi di informazioni diversi:

- Tempo solo invio
- Istruzioni Perf-Event
- Cicli TSC (read\_tsc)
- Tempo invio-ricezione

In particolare nel Publisher 4.2 prima e dopo la chiamata a funzione di write() si sono presi i valori tempo-invio, istruzioni, e TSC, mentre al lato ricevente, di Subscriber 4.2 è stato preso il tempo al momento dell'arrivo del messaggio. Segue uno schema uml della base di ognuno dei test.

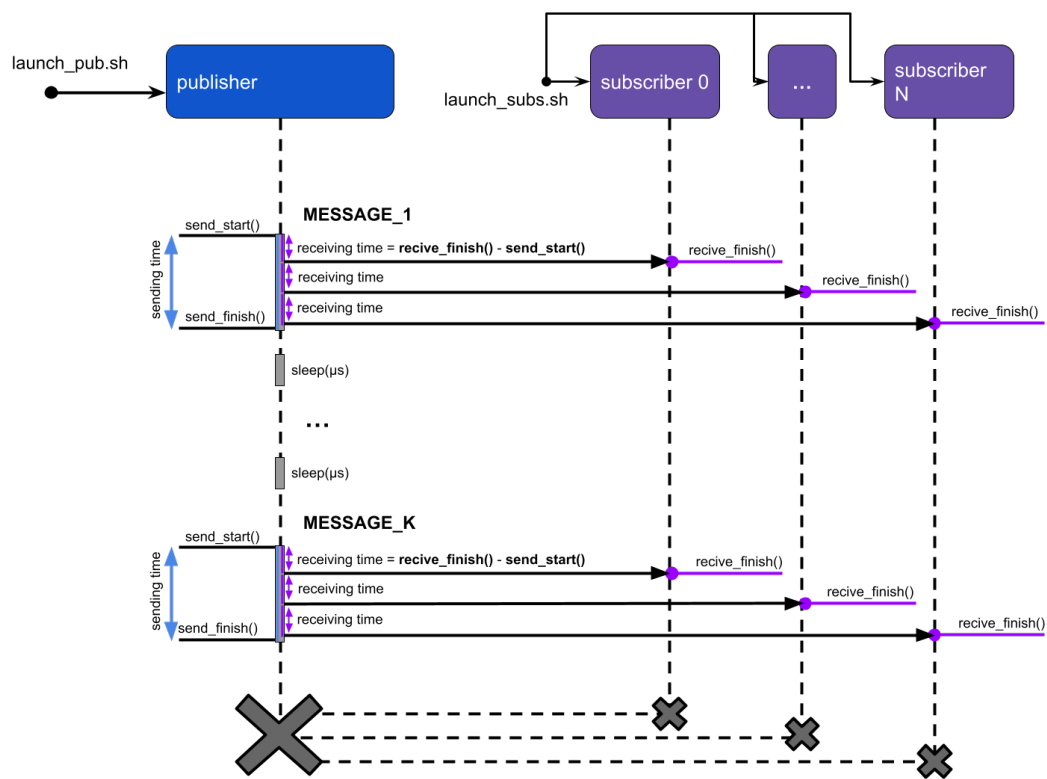
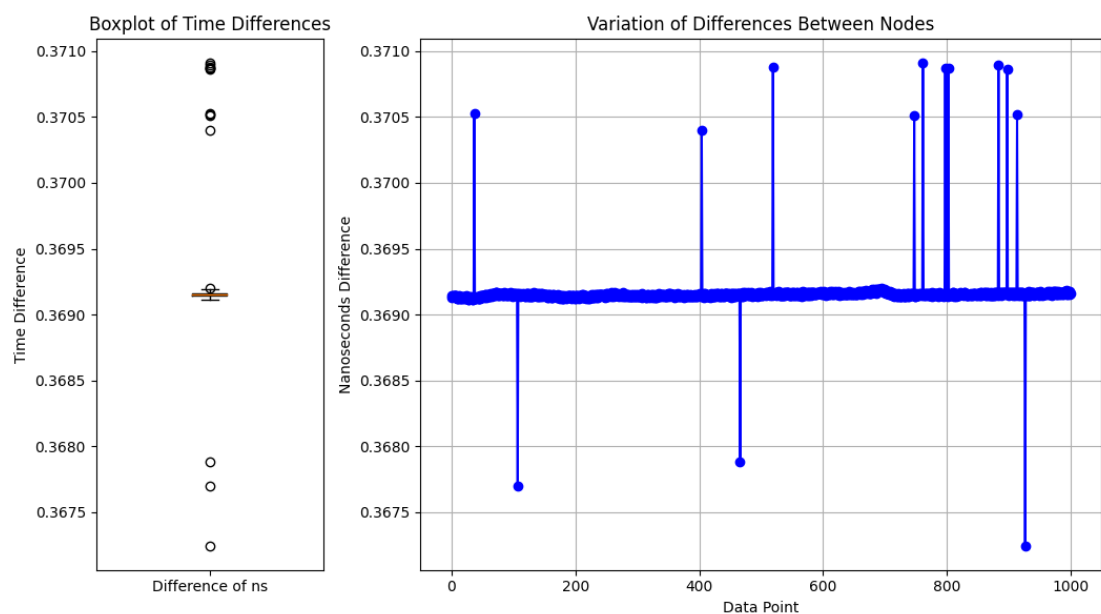


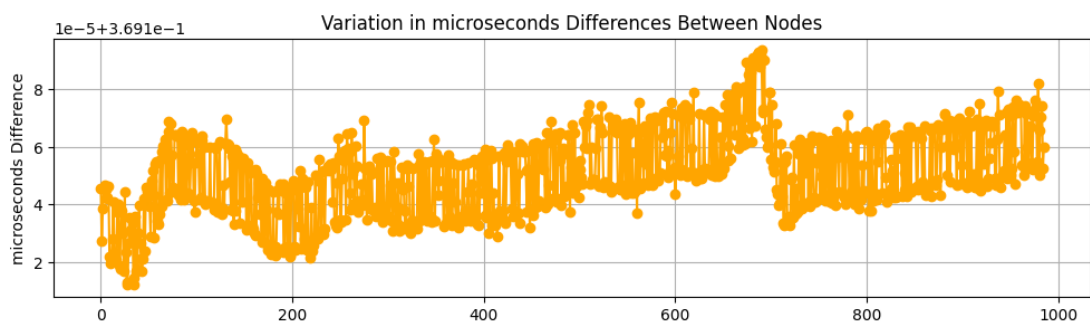
Figura 5.1. Schema UML

### Sincronizzazione orologi su nodi diversi

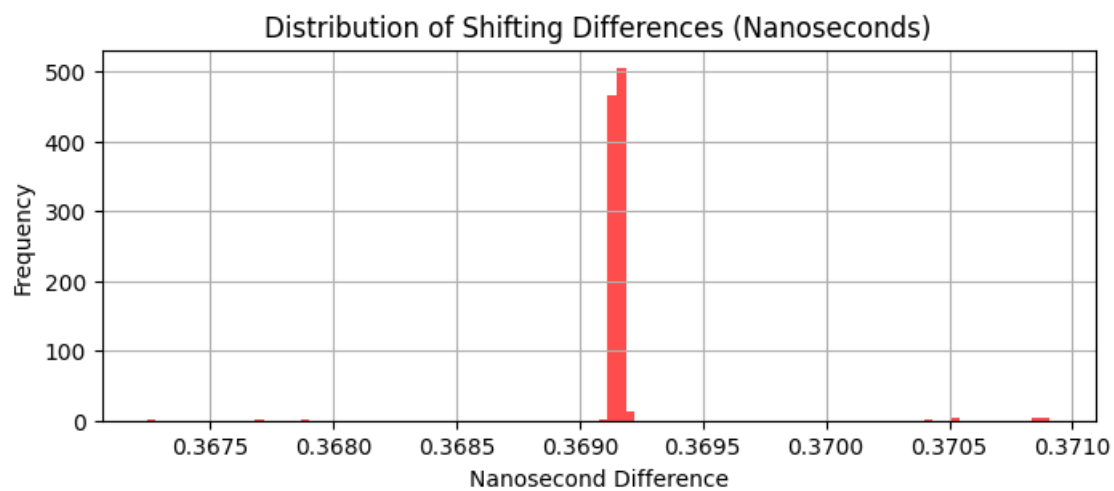
Per ottenere risultati attendibili sulla metrica del tempo è stato necessario sincronizzare i nodi utilizzati prima di poter far partire i test. Per farlo è stata usata una funzione *CLOCK\_MONOTONIC* che rappresenta un *un orologio non impostabile a livello di sistema che rappresenta il tempo monotono da un punto non specificato nel passato*. Su Linux, quel punto corrisponde al numero di secondi di esecuzione del sistema da quando è stato avviato. L'orologio *CLOCK\_MONOTONIC* non è influenzato da salti discontinui nell'ora del sistema, ma è influenzato dalle regolazioni incrementali eseguite da NTP. Il problema che si è presentato, è che avendo nodi diversi su cui far eseguire i test, per provare ad esempio nel modo più affidabile i protocolli di trasporto, è stato necessario implementare delle MPI\_Barrier prima di diverse esecuzioni di *clock\_gettime(CLOCK\_MONOTONIC)*. Di seguito sono stati riportati i grafici del risultato ottenuto.



**Figura 5.2.** Scostamento del tempo su nodi diversi



**Figura 5.3.** Scostamento senza outliers

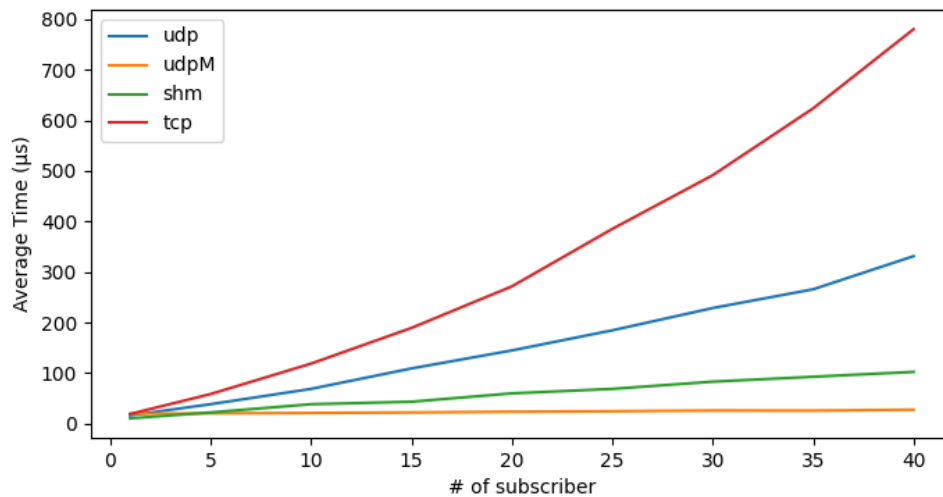


**Figura 5.4.** Distribuzione delle differenze

Come possibile vedere nella figura 5.2 nonostante le `mpi_barrier`, sono presenti degli scostamenti di tempo tra 2 nodi durante diversi test effettuati (in particolare 1000), e si è scelto di utilizzare il valore modale di questa differenza, sulla base del quale, si sono elaborati tutti i dati successivi.

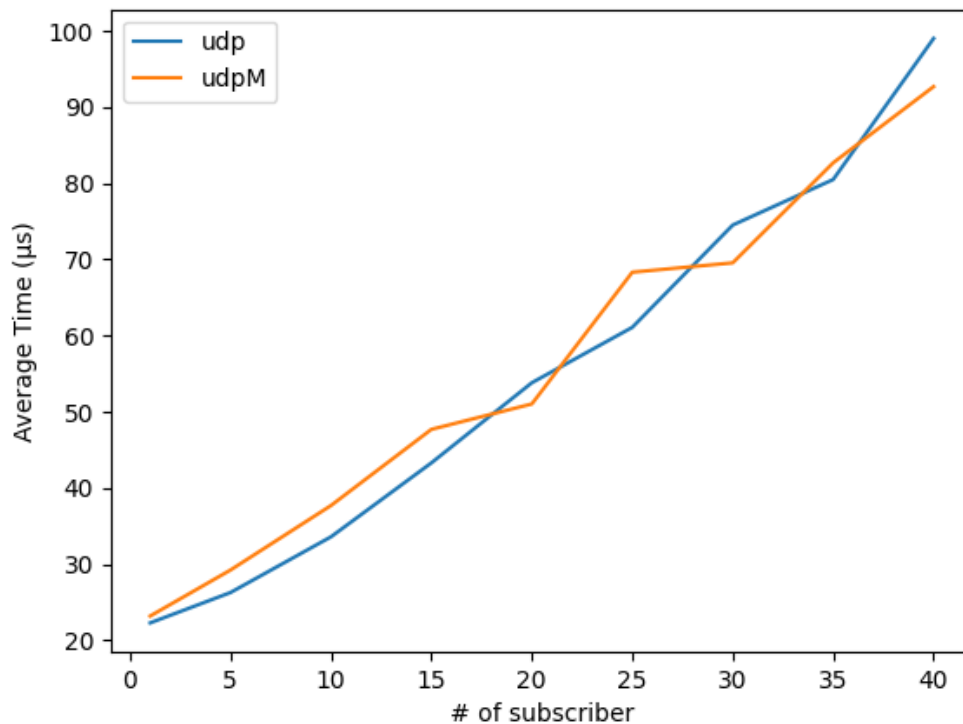
### 5.1.2 Impatto del numero di sub in un dominio

Visto lo schema 5.1 risulta facile capire, che il numero di subscriber presenti in un dominio comporta un overhead di comunicazione che va ad influenzare sia i tempi, che i cicli, che le istruzioni impiegate nella singola *publish* su un topic che viene facilmente dimostrato nella figura 5.5.



**Figura 5.5.** overhead sulla publish all'aumentare dei subscriber

Ovviamente l'impatto è poco significativo in quei protocolli che applicano strutture di multicasting (spiegata successivamente) come udp-Multicast e Shared-Memory.



**Figura 5.6**

Questo può portare una singola publish a impiegare più cicli e più tempo della singola ricezione dei messaggi, come si vede nella figura 5.7

In tutti i test successivi, ove non specificato diversamente sono stati usati 1 publisher e 48 subscriber su diversi nodi. Questo è stato fatto per provare la scalabilità, visto che nel testbed che è stato utilizzato, erano presenti 48 core (1 core per ogni subscriber).

### 5.1.3 Test-1

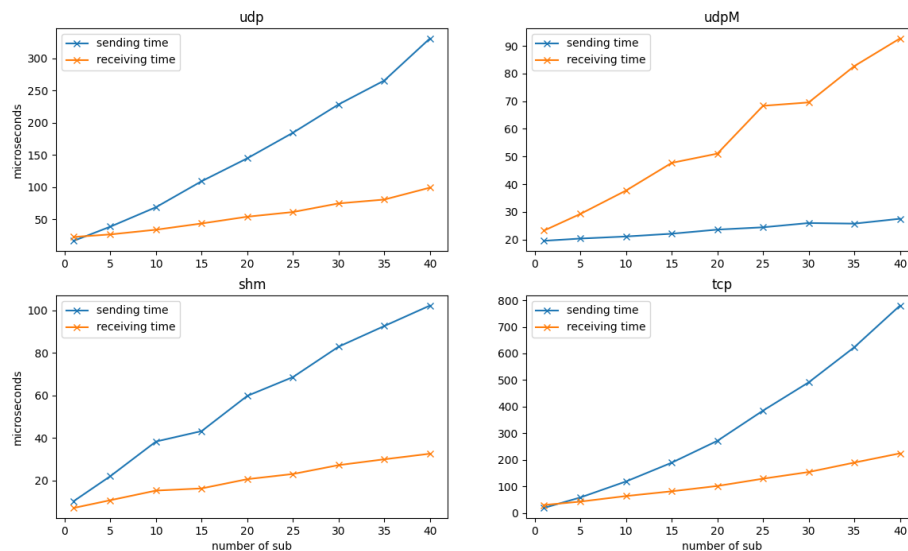
In DDS ed in particolare nel layer sottostante di RTPS, per scambiare messaggi anche tramite rete, e non solo nello stesso nodo, è possibile scegliere come mezzo diversi tipi di protocolli:

- udp: fornisce due versioni v4 e v6 e importa l'omonimo protocollo di trasporto
- tcp: fornisce due versioni v4 e v6 e importa l'omonimo protocollo di trasporto



- **udp-multicast**: una versione modificata del semplice udp, dove tutti i subscriber collegati allo stesso topic, hanno un indirizzo comune di ricezione dei dati, permettendo così al publisher di inviare un singolo messaggio che viene condiviso tra tutti i subscriber
- **shared-memory**: analogo al metodo precedentemente, ma invece di utilizzare un indirizzo IP, viene utilizzato un indirizzo di memoria. E' possibile solo quando i due processi che comunicano sono sullo stesso nodo, con memoria condivisa.

Nel primo test si è valutata la differenza di queste implementazioni utilizzando la rete infiniband 5 su diversi nodi di un supercalcolatore. I risultati che sono stati trovati forniscono importanti informazioni,



**Figura 5.7.** differenza tra solo publish e publish-subscribe per ogni protocollo

### 5.1.4 Test-2

Un concetto fondamentale nelle comunicazioni tra attori con gerarchie diverse, in sistemi con diverse centinaia di migliaia di entità, come cluster, nodi, processori, workflow, job (etc.), sono le possibilità di instradare, segmentare e rendere gerarchiche le comunicazioni. Come spiegato nel capitolo 4 in DDS ci sono diversi strumenti disponibili per farlo. Tra di loro differiscono per alcuni aspetti, come flessibilità, costo (in performance) e livello di segmentazione.

In questo test si è valutata la differenza in termini di performance dei diversi strumenti, con un particolare focus sulle partizioni e le wildcards rese disponibili in esso.

## **Comparazione strumenti**

Nei test effettuati con domini, topic e partizioni, non sono state notate differenze degne di nota in termini di performance (cicli e istruzioni) nell'usare uno strumento piuttosto che un altro.

### **Dominio**

Il dominio è la segmentazione di più "forte" e di più alto livello. Va a partizionare gli attori presenti in un dominio in modo del tutto fisico (cambiando per ogni dominio porte e indirizzi di comunicazione) e per nulla flessibile. Per cambiare il dominio è necessario distruggere e creare di nuovo il partecipante. Inoltre il dominio non permette nessun tipo di gerarchia.

### **Topic**

All'interno di un dominio i topic definiscono il metodo principale di instradamento dei messaggi, essendo però limitato dal tipo di messaggio che si vuole inviare. Infatti topic diversi supportano tipi di dato diversi, e non sono modificabili a run-time. Inoltre il topic non permette gerarchie ed è difficilmente modificabile a run-time

### **Partizione**

Questo strumento risulta molto interessante, in quanto all'interno di un topic permette di definire gerarchie (è possibile sottoscrivere a più partizioni contemporaneamente), definisce wildcards e crea una segmentazione virtuale. Inoltre è facilmente modificabile a run-time.

### **Wildcards**

Le wildcards sono un costrutto appartenente alle partizioni, che permette di definire dei pattern testuali sulla base del quale vengono instradati i messaggi. Un esempio può essere *Node\** che va a corrispondere a tutti i messaggi sotto il topic precedentemente definito, a tutte le partizioni che iniziano con Node.

### **5.1.5 Test-3**

## **5.2 Risultati**

## **5.3 Modello**

## **5.4 Scheletro componenti**

Nel seguente capitolo viene stilato uno scheletro dei componenti con i relativi topic usati al fine di dare una visione completa e aggiuntiva rispetto al modello precedentemente stilato

DUMMIES	
NAME	USED
NODE MANAGER DUMMY	<ul style="list-style-type: none"> <li>● P 0 default Monitor_report_job_telemetry</li> <li>● P 0 default Monitor_report_node_telemetry</li> <li>● P 0 default Monitor_report_cluster_telemetry</li> </ul>
JOB SCHEDULER DUMMY	<ul style="list-style-type: none"> <li>● P 0 default SystemPowerManager_get</li> <li>● P 0 default SystemPowerManager_set</li> <li>● S 0 default SystemPowerManager_get_reply</li> <li>● S 0 default SystemPowerManager_set_repl</li> </ul>
JOB MANAGER DUMMY	<ul style="list-style-type: none"> <li>● S 0 default NodeManager_get</li> <li>● P 0 default NodeManager_get_reply</li> </ul>
SERVERS	
NAME	OFFERED
NODE MANAGER	<ul style="list-style-type: none"> <li>● S 0 default NodeManager_get</li> <li>● S 0 default NodeManager_set</li> <li>● P 0 default NodeManager_get_reply</li> <li>● P 0 default NodeManager_set_reply</li> </ul>
SYSTEM POWER MANAGER	<ul style="list-style-type: none"> <li>● S 0 default SystemPowerManager_get</li> <li>● S 0 default SystemPowerManager_set</li> <li>● P 0 default SystemPowerManager_get_reply</li> <li>● P 0 default SystemPowerManager_set_reply</li> </ul>
MONITOR	<ul style="list-style-type: none"> <li>● S 0 default Monitor_report_job_telemetry</li> <li>● S 0 default Monitor_report_node_telemetry</li> <li>● S 0 default Monitor_report_cluster_telemetry</li> </ul>

# Conclusioni

È stato dimostrato come un framework di comunicazione DDS può essere usato all'interno di un Power-Stack per la gestione di energia in sistemi HPC vincolati dalla potenza al fine di affrontare il problema della limitazione energetica.

# Bibliografia

- [1] TODO. *TODO*. 2023. URL: <https://wikipedia.it>.
- [2] INTEL. *GEOPM*. 2017. URL: [https://sc17.supercomputing.org/SC17%20Archive/tech\\_poster/poster\\_files/post176s2-file3.pdf](https://sc17.supercomputing.org/SC17%20Archive/tech_poster/poster_files/post176s2-file3.pdf).
- [3] Daniel Hackenberg et al. «HDEEM: High Definition Energy Efficiency Monitoring». In: *2014 Energy Efficient Supercomputing Workshop*. 2014, pp. 1–10. DOI: [10.1109/E2SC.2014.13](https://doi.org/10.1109/E2SC.2014.13).
- [4] CSLab;NTUA. *Open Architecture for Future Supercomputers*. 2021. URL: <https://regale-project.eu/> (visitato il 07/05/2023).
- [5] Object Management Group. *Data Distribution Service*. 2004. URL: <https://www.omg.org/spec/DDS/1.0>.
- [6] Object Management Group. *DDS Interoperability Wire Protocol*. 2008. URL: <https://www.omg.org/spec/DDSI-RTPS/2.0>.
- [7] eProsima. *FastDDS*. 2022. URL: <https://fast-dds.docs.eprosima.com/en/v2.11.2/>.
- [8] Giacomo Madella. *github/tesiMagistrale*. 2023. URL: <https://github.com/madella/tesiM>.