# RE.VE.N.GE. DDS con Fault-tolerance del Sistema di Consegna

Marco Livini – marco.livini@studio.unibo.it

Gruppo: Nardelli, Marco Livini, Christian Pinto

***Abstract***

***Il mondo legato ai servizi di distribuzione dell'informazione è in forte trasformazione. Uno degli elementi che stanno guidando questa trasformazione è il nuovo modo con cui tali servizi possono essere ripensati attraverso soluzioni ed architetture convergenti su reti IP. La diffusione di Internet ha permesso di trasformare, ottimizzare, arricchire e, addirittura creare nuove modalità con cui tali servizi oggi vengono erogati e fruiti. Lo scopo di questo lavoro è la definizione di un sistema middleware, basato sullo standard OMG Data Distribution Service, di supporto per la distribuzione di notizie su larga scala da parte di agenzie di stampa, con particolare attenzione agli aspetti di tolleranza ai guasti e sincronizzazione.***

## I. Introduzione

La capillare diffusione di Internet ha proposto nuovi scenari di distribuzione dell'informazione, con caratteristiche profondamente diverse da quelle a cui i classici canali di distribuzione ci avevano abituati. In particolare la "società di Internet" dispone oggi di servizi la cui fruibilità risulta essere indipendente dalla localizzazione geografica e non vincolata a particolari finestre temporali predeterminate, diversamente da ciò che avviene ad esempio con la stampa o la tv.

Lo scopo di questo lavoro è la definizione di un nuovo sistema middleware di supporto per la distribuzione di notizie su larga scala. In particolare si esplorerà la possibilità di realizzare un sistema di tal genere avvalendosi di un implementazione commerciale dello standard *OMG Data Distribution Service* prodotta dalla Real-Time Innovations.

In particolare si procederà descrivendo in primo luogo il modello publish-subscribe e lo standard *OMG Data Distribution Service* per poi descrivere brevemente le caratteristiche dell'implementazione prodotta della Real-Time Innovations. In seguito verrà presentata l'architettura scelta per la realizzazione del sistema, esplicandone i concetti fondamentali cercando per quanto possibile di motivare le scelte architetturali fatte, mettendole in relazione con le problematiche imposte dal conteso di in cui il sistema si troverà ad operare. Per poi passare, a quella che rappresenta la parte più corposa di questo lavoro e cioè, la gestione della tolleranza ai guasti e della sincronizzazione tra le repliche. Infine verranno proposti alcuni risultati sperimentali, ottenuti testando il sistema in ambiente distribuito e confrontati con i risultati teorici attesi, quali ad esempio la velocità del sistema nel riconfigurarsi a fronte della caduta di alcuni nodi, o la velocità con cui una nuova copia del sistema viene aggiunta al set di quelle precedentemente disponibili e sincronizzata con queste ultime, etc.

## II. Il modello Publish-Subscribe

Dovendo realizzare un sistema middleware di supporto per la distribuzione di notizie, il modello publish-subscribe si candida ad essere una promettente soluzione a molte delle principali problematiche di comunicazione imposte da un sistema di tal genere. Esso realizza un pattern di comunicazione che potremmo definire ad "accoppiamento lasco" che punta alla realizzazione del completo disaccoppiamento nel tempo, nello spazio e nella sincronizzazione. Il modello di distribuzione dei dati è basato sulla nozione di evento, inteso come variazione di stato di una delle entità facenti parte del sistema.

Si possono distinguere diverse tipologie di evento, un evento può essere tipo:

- **Primitivo:** privo di alcun contenuto informativo generato come semplice notifica del cambiamento di stato di una qualche entità del sistema.
- **Con contenuto:** portano con se del contenuto informativo, tipicamente indicante alcune informazione riguardati i motivi e le responsabilità della sua generazione.
- **Con qualità:** offrono la possibilità di specificare in maniera differenziata per i diversi utilizzatori alcune caratteristiche riguardanti ordine di arrivo, reliability, persistenza, tempo massimo di mantenimento, priorità, etc.

Il modello publish/subscribe rappresenta un paradigma di interazione che vede coinvolte due entità: produttori (*publisher*) e consumatori (*subscriber*, o sottoscrittori). Consente ai sottoscrittori di esprimere i propri interessi verso un certo tipo di evento o un pattern di eventi, con lo scopo di essere successivamente notificati con eventi compatibili con gli interessi espressi. In altre parole, i produttori pubblicano informazioni su un canale di eventi e i sottoscrittori si sottoscrivono al canale per ricevere informazioni di loro interesse. Il modello base del sistema per l'interazione publish/subscribe si basa su

un servizio di notifica di eventi (*event notification service*, o più semplicemente *event service*), il quale memorizza e gestisce le sottoscrizioni e consegna gli eventi. L'*event service* funge da mediatore tra i publisher, che svolgono la funzione di produttori di eventi, e i subscriber, che svolgono la funzione di consumatori di eventi. I subscriber registrano i propri interessi verso un certo tipo di evento senza essere a conoscenza dell'effettiva sorgente dell'evento.

Il punto di forza di questo modello è il disaccoppiamento (*decoupling*) che l'*event service* crea tra publisher e subscriber, rendendo il modello estremamente flessibile. Questa proprietà si esplica su tre livelli:

- **Spaziale:** i publisher e i subscriber non hanno necessità di conoscersi a vicenda. I publisher pubblicano eventi attraverso l'*event service* e i subscriber li ricevono da quest'ultimo. I publisher non mantengono un riferimento ai subscriber, né sanno quanti subscriber sono presenti nel sistema. Lo stesso vale per i subscriber.
- **Temporale:** non occorre che le parti che interagiscono (publisher e subscriber) siano attive nello stesso momento.
- **Sincronizzazione:** non esiste sincronizzazione tra publisher e subscriber, quindi i publisher, all'atto della produzione di un evento, non restano in attesa che un subscriber interessato venga notificato; analogamente i subscriber ricevono, in modo asincrono, una notifica da parte dell'*event service* quando l'evento a cui sono interessati è disponibile.

I subscriber, in generale, sono interessati solo ad un sottoinsieme di eventi prodotti dai publisher. I diversi modi di specificare gli eventi di interesse ha condotto a differenti schemi di sottoscrizione.

Tre sono i principali schemi utilizzati:

- **Topic-based**: basato sulla nozione di *topic*. I partecipanti all'interazione possono pubblicare eventi e sottoscriversi a specifici *topic*, identificati da una *keyword*. Ciascun *topic* rappresenta un differente canale di comunicazione.
- **Type-based**: in questo schema la sottoscrizione viene effettuata sulla base del tipo di evento, ovvero la sua struttura stessa dell'evento.
- **Content-based:** supera gli svantaggi dei modelli discussi precedentemente. Gli eventi non sono più classificati in base a criteri prestabiliti (ad esempio il nome di un topic) ma in base alle loro proprietà, come ad esempio attributi interni alla struttura dati che rappresenta l'evento o addirittura metadati associati all'evento.

## III. Data Distribution Service

### *A. Lo standard OMG Data Distribution Service*

Lo standard *OMG DDS* è la definizione di un'architettura di tipo *Model Driven Architecture* specifica un'infrastruttura indipendente dalla
Piattaforma su cui è possibile derivare un modello che sia inerente alle specifiche della propria applicazione e della propria piattaforma. Il *Data Distribution Service* fonde un'architettura basata sul paradigma appena descritto, con la nozione di qualità del servizio. Ogni entità del sistema può specificare i requisiti necessari al proprio corretto funzionamento, e in questo modo il servizio può definire (se possibile) un'allocazione efficiente delle risorse . Questa caratteristica rende i middleware basati su *DDS* una soluzione da adottare laddove il sistema richiedono caratteristiche stringenti in termini di efficienza, determinismo, flessibilità e supporto per la realizzazione della fault-tolerance.

Il *Data Distribution Service* realizza un paradigma di tipo data-centrico in cui l'attenzione è rivolta alla definizione e caratterizzazione dei dati, soprattutto per ciò che concerne la qualità di servizio. La principale implicazione di questo concetto è il fatto che l'instradamento della comunicazione avviene in base ai dati che devono essere trasferiti, ed in base alla qualità di servizio per essi richiesti.

Per far ciò lo standard *DDS* adotta un modello di interazione publish-subscribe di tipo topic-based; ad ogni tipo di dato è associato un topic.

Lo standard DDS definisce due livelli:

- **DCPS** (*Data-Centric Publish/Subscribe*): fornisce le funzionalità utili ad un'applicazione per la pubblicazione e la sottoscrizione dei dati, oltre alla gestione del trasporto *efficiente* alle destinazioni.
- **DLRL** (*Data Local Reconstruction Layer*): fornisce API di accesso e di filtro per la manipolazione dei dati ricevuti dal livello DCPS.

Il *Data Distribution Service* permette di adottare una serie di politiche di qualità del servizio in modo da poter adattare il servizio alle caratteristiche e ai requisiti dell'applicazione. Esse sono un insieme di proprietà che guidano il comportamento del servizio e sono realizzate a partire da una policy predefinita che viene specializzata per definire i diversi aspetti del funzionamento del servizio, quali ad esempio reliability, tempi massimi di validità dei dati, ownership, ect .

### *B. L'implementazione Real-Time Innovations*

L'implementazione DDS della RTI implementa in maniera integrale lo standard DDS, introducendo inoltre alcune estensioni proprietarie. Offre supporto per la maggior parte dei sistemi operativi e i linguaggi di programmazione oggi più diffusi. Essa rappresenta un middleware composto da una libreria run-time, un database e una serie di processi. L'applicazione è linkata alla libreria, mentre dei processi demone

realizzano le diverse funzioni del middleware, come l'inoltro dei messaggi, il marshaling e unmarshaling dei parametri. I demoni sono, inoltre, responsabili della condivisione e dell'aggiornamento del database locale, della gestione delle sottoscrizioni, dell'invio e della ricezione di nuove pubblicazioni.

Per l'implementazione è stata scelta un'architettura completamente distribuita di tipo peer-to-peer, priva di qualunque punto singolo di fallimento.

## IV. REliable and VErsatile News delivery support for aGEncies

Il progetto RE.VE.N.GE. si propone di sviluppare un sistema middleware di supporto per la distribuzione di notizie su larga scala da parte di agenzie di stampa.

Il sistema che dovrà essere realizzato dovrà garantire elevata affidabilità del sistema di consegna. In uno scenario di tal genere possiamo individuare tre principali attori:

- **Fonti:** Rappresentano le entità che generano le informazioni.
- **Fruitori:** Rappresentano i destinatari finali delle informazioni generate dalle fonti, essi selezioneranno in base alle proprie preferenze gli argomenti delle news che desiderano ricevere.
- **Sistema di consegna:** Esso dovrà acquisire le informazioni generate dalle fonti ed in seguito smistarle ai fruitori in base alle loro preferenze. Il tutto dovrà, ovviamente, essere fatto in maniera tale da rispettare eventuali vincoli di qualità di servizio richiesti.

### *A. Assunzioni*

Per la realizzazione del sistema RE.VE.N.GE. sono state fatte alcune assunzioni di base a partire da alcune considerazioni riguardo al campo applicativo del sistema che si sta progettando. In particolare si assume che il numero totale delle Fonti sia complessivamente molto inferiore al numero totale dei Fruitori. E che i link di rete che collegano le Fonti e i Fruitori al sistema siano tali da non supportare, in generale, comunicazione di tipo multicast.

### *B. Architettura*

L'architettura di base del sistema di consegna delle notizie prevede che il sistema sia ridondato su una molteplicità di nodi. Tutte le repliche del sistema sono copie attive, questo per suddividere il carico di gestione dei fruitori tra i vari nodi partecipanti alla realizzazione del sistema complessivo. Ognuno di queste repliche riceve le notizie da tutte le fonti, e ad essa è delegato la gestione di una quota dei fruitori che si sono registrati al sistema (Figura 1); in particolare ogni fruitore viene assegnato ad una partizione, e ad ogni copia del sistema viene assegnata la gestione di una o più partizioni.

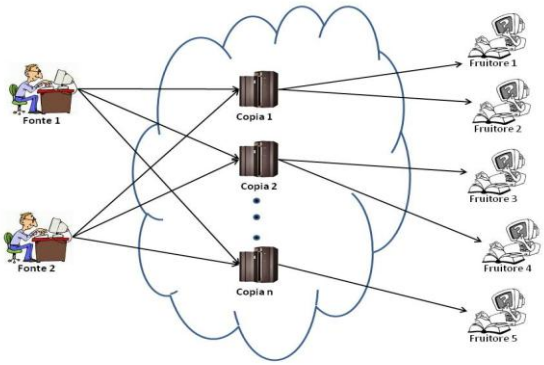


Figura 1. Architettura di Base del sistema di consegna.

Un architettura siffatta sebbene si presti bene a distribuire il carico sul fronte dei fruitori, presenta alcuni limiti di scalabilità. Questi sono dovuti al numero di repliche introducibili nel nostro sistema di notifica che deve, comunque, non superare certi limiti; poiché al crescere del numero delle copie cresce proporzionalmente il numero di invii che devono essere effettuati dalle fonti.

Per questo motivo l'architettura di base è stata ampliata introducendo il concetto di località e di conseguenza delle nuove entità capaci di gestire il nuovo scenario. In particolare è stato deciso di realizzare un'architettura multi dominio. In particolare ogni dominio è rappresentato da un sistema di consegna di base come quello appena visto, affiancato da un componente, chiamato *Relay*, che si occupa di redirigere le news generate nel dominio corrente verso gli altri domini ed acquisire le news prodotte dagli altri domini per poi immetterle nel sistema di consegna del dominio corrente (Figura 2).

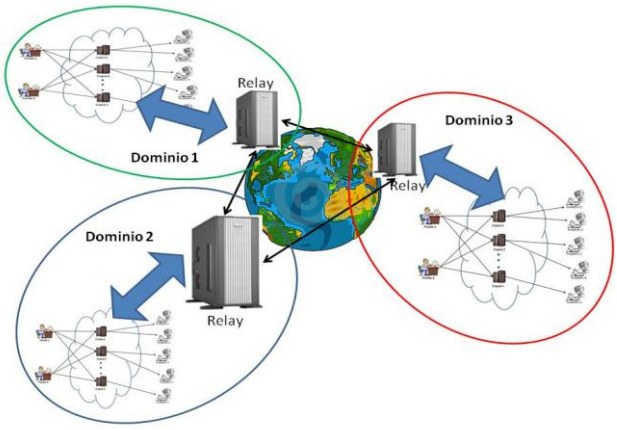


Figura 2. Architettura multi dominio del sistema di consegna.

Grazie a questa nuova strutturazione del nostro sistema si ottiene un maggiore livello di scalabilità, ed al contempo una maggiore efficienza nella distribuzione delle news locali. Per maggiori dettagli e considerazioni sull'architettura del sistema di consegna sviluppato si faccia riferimento al lavoro prodotto da Luca Nardelli.

### C. *Supporto alla Fault Tolerance*

Uno dei requisiti fondamentali per un sistema distribuito su larga scala è la capacità del sistema stesso di far fronte ai guasti, cercando per quanto possibile di mantenere il servizio fornito attivo e con livelli di qualità accettabili.

Con lo scopo di raggiungere tale obbiettivo è stata introdotta una forma di replicazione per ognuna delle componenti del sistema di consegna, in particolare è stata prevista una forma di replicazione a copie attive per il sistema di consegna vero e proprio e una forma di replicazione master-replica per i relay. Per l'attuazione dei protocolli di sincronizzazione delle copie sono stati utilizzati dei canali di comunicazione DDS ad hoc (creazione di un topic di controllo) di tipo reliable e con garanzia sull'ordine di arrivo dei messaggi.

#### i. *Il concetto di failure*

Prima di passare all'analisi dettagliata della gestione della fault-tolerance è bene chiarire in che modo una copia sia del sistema di consegna che del sistema di Relay viene dichiarata fallita e non più attiva.

Per rilevare questo tipo di eventi è stato scelto, dopo un'attenta analisi del middleware DDS offerto dalla Real-Time Innovations, di utilizzare le notifiche di *liveness_change* generate dal middlware di rete per identificare una variazione del insieme dei partecipanti all'interazione per un determinato topic.

In particolare, partendo dal presupposto che ogni copia del sistema ha a sua disposizione un entità publisher ed un entità subscriber dedicata alla gestione dei messaggi di controllo, si è deciso di utilizzare le notifiche di *liveness_change* che tutti i subscriber ricevono a fronte di una variazione dell'insieme dei publisher partecipanti all'interazione. Tali notifiche vengono automaticamente generate dal middleware di rete, che implementa un meccanismo basato sull'invio ad intervalli di tempo periodici di *heartbeat.*

La scelta degli intervalli di tempo a cui i vari publisher inviano al middlware gli heartbeat può essere opportunamente settato per ottenere da parte del sistema la reattività richiesta. Nel nostro caso specifico l'invio di questi heartbeat è cadenziato ad intervalli regolari di di 800 millisecondi, metre la rilevazione è effettuata ogni 2400 millisecondi. La scelta di questi valori è stata effettuata a partire dai risultati forniti da alcune prove pilota del sistema su una rete sottoposta a congestione.

#### ii. *Replicazione del sistema di consegna*

Per il sistema di consegna, come già preannunciato, è stato introdotto un sistema di consegna a copie attive questo con il preciso scopo di ottenere oltre che maggiore affidabilità del sistema anche maggiori performance complessive in termini di numero massimo di fruitori gestibili.

Ad ogni copia del sistema viene automaticamente assegnato un identificativo "univoco" utilizzando l'algoritmo di generazione degli UUID.

All'avvio del sistema le varie copie in modo automatico, o in base ad una politica decisa a priori dall'utente, eleggono un master e si scambiano reciprocamente delle informazione per l'inizializzazione.

Il nodo master, in fase si avvio, inizializza una struttura dati, chiamata *partitionMap*, nella quale verranno, via via, memorizzate informazioni riguardanti tutte le copie facenti parte del sistema; in particolare per ogni copia (master compreso) si mantiene la lista delle partizioni gestite e per ognuna di queste partizioni il numero di fruitori attualmente fancenti parte della stessa. In seguito ogni copia che vuole aggiungersi al sistema invia un messaggio di *Join* (Figura 3) a tutte le altre repliche indicando il suo identificativo. L'attuale copia master, a questo punto invia una copia della propria *partitionMap* alla copia che ha generato il messaggio di *Join.* In seguito la copia invia a tutti un messaggio di notifica, *MapOK,* la corretta ricezione della *partitionMap*, dopo questa operazione la sua fase di inizializzazione è completa. Ogni copia appena riceve un messaggio di *MapOK* aggiorna la propria *partitionMap* con una nuova entry riguardante la nuova copia. L'invio del messaggio di *Join* a tutte le copie del sistema, benché possa apparire futile ad una più attenta analisi, risulta essere di fondamentale importanza per far fronte a situazioni in cui il nodo master "cada" durante le procedure di *Join* di una nuova copia; infatti in questi casi il nodo che verrà eletto come nuovo master potrà ultimare la procedura di inizializzazione del nodo che ha generato la richiesta di *Join.*

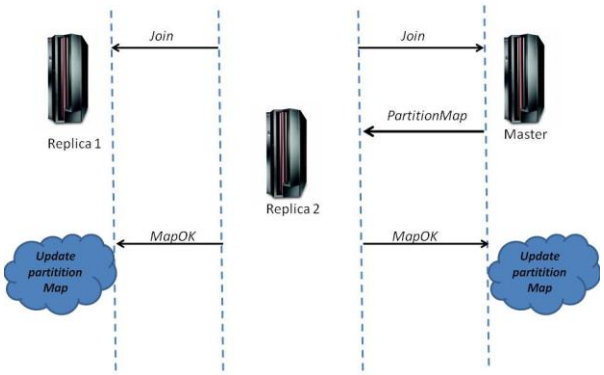


**Figura 3. Protocollo di Join**

Terminata la fase di inizializzazione ogni copia riceve tutte le news generate da tutte le fonti del dominio di appartenenza e le inoltra verso i fruitori di propria competenza; oltre a fare cioè ne salva una copia in una coda temporanea (la cui dimensione massima è determinata dall'utente in fase di deploy), questo come vedremo in seguito sarà utile per evitare che i fruitori perdano delle news nel momento in cui la replica che li sta servendo "cade", rendendo così la *failure* completamente, o quasi, trasparente ai fruitori.

Ogni qual volta un nuovo fruitore richiende di sottoscrivere il servizio di ricezione di news invia un messaggio a tutte le repliche. Ognuna di queste repliche invia a tutte le altre un messaggio di offerta (Figura 4) in cui indica una stima, ottenuta secondo una qualche politica di bilanciamento, del proprio carico. In questo modo, ognuna delle copie può aggiornare la propria *partitionMap* allocando il fruitore alla replica con il carico più basso. Durante questa procedura, vi è, ovviamente, la possibilità che una delle repliche "cada", al verificarsi di questa condizione il middlware DDS genera un evento di *liveness_changed* che reinnesca il processo di offerta e allocazione del fruitore in maniera completamente trasparente al fruitore stesso.

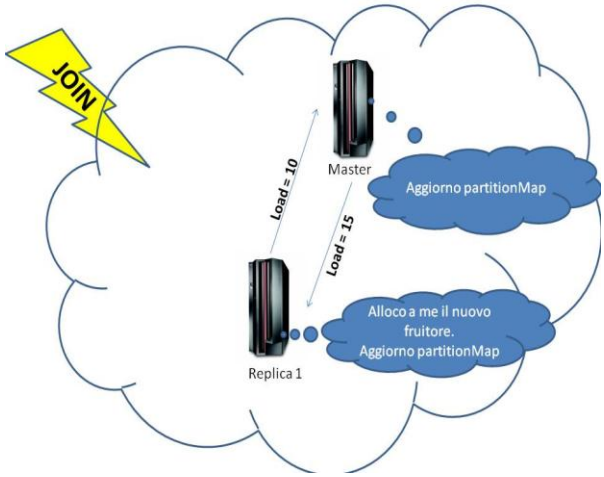


**Figura 4. Procedura di allocazione nuovo fruitore.**

Una volta a regime le varie repliche notificano a tutte le altre repliche il loro stato rispetto all'invio di news, indicando quale è l'ultima news correttamente consegnata a tutti i fruitori interessati. La cadenza, in termini di numero di news smistate, con cui queste notifiche devono essere generate è un parametro che deve essere specificato a tempo di deploy per ciascuna istanza del sistema. Tuttavia è bene tenere in considerazione che valori molto piccoli per questo parametro possono condizionare pesantemente le prestazioni del sistema introducendo un alto *overhead* di comunicazione tra le copie.

Quando si verifica una condizione di *failure*, come abbiamo già avuto modo di dire, il middlware DDS genera un evento di *liveness_changed* (Figura 5), il contenuto dell'evento generato, non da alcuna indicazione sul nodo che ha subito la *failure*, anche se riesce comunque a dare informazioni riguardo al numero complessivo di nodi ancora in stato di *live*. Per questo motivo al verificarsi di questo evento ognuna delle repliche attualmente facenti parti del nostro sistema si occupa di inviare a tutte le altre repliche un messaggio di *LIVE*, indicando anche una stima del proprio carico. Localmente ogni replica ricevuti i segnali di *LIVE* da tutte le altre repliche ancora attive aggiorna la propria *partitionMap* rimuovendo le informazioni di stato dei nodi non più attivi ed allocando le partizioni di questi alla replica attualmente meno carica, in base alle stime fornite tramite il messaggio di *LIVE* da ciascuna replica ancora attiva.

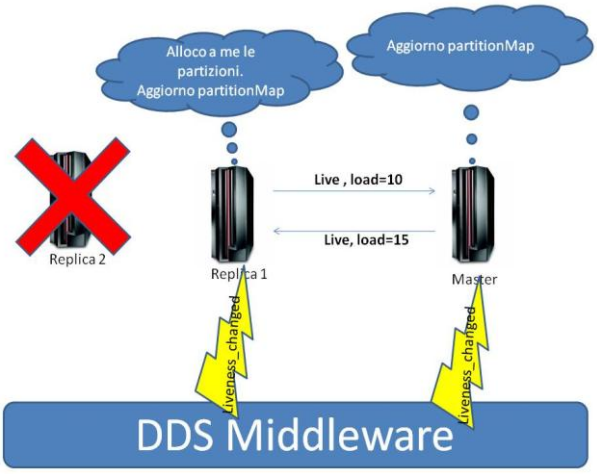


**Figura 5. Riconfigurazione del sistema di consegna a fronte della failure di una copia.**

Anche in questo caso qualora durante il processo di riconfigurazione del sistema si verifiche una nuova *failure* il processo di aggiornamento della *partitionMap* e riallocazione delle partizioni verrà re inizializzato automaticamente.

Ovviamente questo surplus di carico risulterà essere solo una condizione transitoria che si risolverà non appena la replica che ha subito la *failure* terminerà il recovery, oppure un'altra replica farà si aggiungere (*Join*) al sistema; infatti in entrambi i casi le partizioni in eccesso allocate ad alcune delle repliche verranno assegnate ai nuovi entranti.

Appena sarà terminata la procedura di aggiornamento dello stato delle copie attive e saranno riallocate le partizioni appartenenti alla replica, o più in generale alle repliche che hanno subito la *failure* bisognerà fare in modo che i fruitori allocati alle suddette partizioni non rilevino, per quanto possibile, il guasto subito dal sistema di consegna. Per fare ciò la repliche che prende in gestione le partizioni "orfane" dovrà occuparsi di inviare a tutti i fruitori, allocati in queste, tutte le news il cui invio non era stato notificato dal sistema "caduto". Per fare ciò, come avevamo già accennato precedentemente, essa dovrà prelevare le news dalla coda temporanea delle vecchie news ed inviare ai fruitori, conformemente alle loro sottoscrizioni, tutte quelle non ancora notificate.

Risulta evidente che in questo modo non esista alcuna garanzia sul fatto che, in talune circostanze, i fruitori non ricevano delle news replicate a meno che in fase di deploy del sistema venga specificato come intervallo di notifica delle news inviate il valore 1, introducendo però, in questo modo, un notevole *overhead* di comunicazione.

Similmente, non è possibile garantire che in ogni situazione nessuna news venga persa, in quanto questo presupporrebbe una coda per le vecchie news di lunghezza infinita, e questo ovviamente risulta inattuabile. Tuttavia un tuning adeguato, basato sulla stima della massima rate di ricezione di news e sulla

durata massima della procedura di aggiornamento delle repliche attive, della la massima lunghezza di questa coda è possibile rendere la probabilità di perdita di news molto bassa.

*iii. Replicazione dei Relay*

Per quanto concerne la replicazione del componente di *Relay* è stato scelto di utilizzare un modello master-replica a copie calde. La scelta di tale modello è dovuta ad alcune considerazioni fatte sullo scenario di utilizzazione del componente.

Come abbiamo già avuto modo di dire il componente di *Relay* è stato pensato come strumento di supporto per il partizionamento geografico del sistema di consegna basato sulla separazione in domini. Inoltre, per il *Relay*, l'assunzione che le fonti siano in numero molto minore rispetto al numero di fruitori non vale più; infatti le fonti per un *Relay* sono tutte le fonti locali e gli altri *Relay* mentre i fruitori sono tutte le repliche del sistema locale e gli altri *Relay*.

In questo scenario un *Relay* assume i connotati di semplice passa-carte che molto probabilmente dovrà limitarsi a prelevare le news ricevute su un interfaccia di rete ed instradarle su di un'altra, in questa ottica l'utilizzo di copie attive può rilevarsi non solo superfluo ma addirittura controproducente a causa dello *overhead* dovuto al mantenimento della sincronizzazione dello stato delle varie copie.

Come nel caso, precedente, della replicazione del sistema di notifica, ad ogni copia del sistema viene automaticamente assegnato un identificativo "univoco" utilizzando l'algoritmo di generazione degli UUID.

All'avvio del sistema le varie copie in modo automatico, o in base ad una politica decisa a priori dall'utente, eleggono un master.

Ognuna delle copie mantiene una lista delle copie disponibili ordinata lessico graficamente in base all'identificativo del sistema, questo perché se il master subisce una *failure*, sarà possibile eleggere il nuovo master senza ulteriori scambi di informazioni tra le copie.

La procedura di Join di una nuova copia, questa vola molto più semplice, prevede l'invio da parte della copia interessata ad entrare a far parte del sistema di Relay di un messaggio di *Join.* In seguito ogni copia attualmente disponibile del sistema di Relay, alla ricezione del messaggio di *Join* risponderà con un messaggio di *Hello* riportante il proprio identificativo e contemporaneamente aggiornerà la propria lista delle copie disponibili. Ricevuti tutti i messaggi di *Hello*, la nuova replica inizializzerà la propria lista delle copie disponibili con gli identificativi delle copie dalle quali ha ricevuto il messaggio di *Hello* (Figura 6).

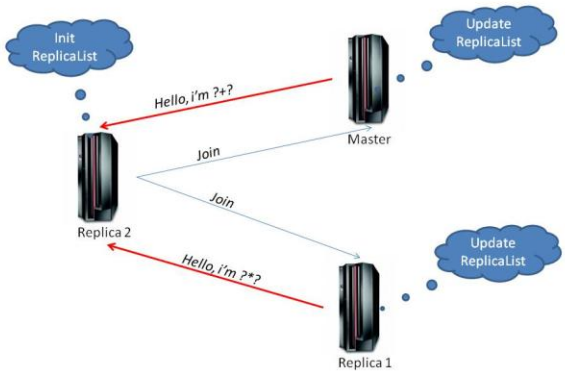


**Figura 6. Join nuova replica del Relay.**

Al termine di questa procedura la copia risulta essere inizializzata. Nel caso in cui una delle repliche "cada" durante questa procedura, in maniera del tutto automatica verrà riavviato il processo di invio dei messaggi di *Hello* da parte delle repliche rimaste ancora attive e di inizializzazione della lista delle copie disponibili della replica che ha richiesto il *Join.*

Dopo l'inizializzazione tutte le repliche ricevono tutte le news sia dal dominio corrente, sia da tutti gli altri domini di Relay. Questa volta però, è solo il master a smistare le news in uscita, mentre le altre copie si limitano ad aggiornare una coda temporanea (la cui dimensione massima è determinata dall'utente in fase di deploy) con le ultime news ricevute, questo come nel caso del sistema di notifica, precedentemente discusso, sarà utile per evitare che vengano perse delle news nel momento in cui la copia master "cade".

Anche nel caso dei Relay per tenere sincronizzato lo stato della copia master con quello delle repliche è stato necessario prevedere un meccanismo di notifica di avvenuta gestione della news, infatti, proprio come nel caso del sistema di consegna, la copia master del Relay periodicamente invia a tutte le copie un messaggio di notifica indicante l'ultima news correttamente gestita. Per quanto riguarda l'intervallo di generazione delle notifiche esso deve essere settato in fase di deploy, e per la determinazione del valore appropriato per questo paramentro valgono le considerazioni fatte precedentemente per il caso del sistema di consegna.

Similmente al caso del *Join* di una nuova copia nel momento in cui una delle repliche "cade", e quindi il middleware DDS notifica l'accaduto con un evento di *liveness_changed*, ogni copia invia a tutte le altre copie un messaggio di *LIVE* con l'indicazione del proprio identificativo (Figura 7). In questo modo ogni replica può ricostruirsi la propria lista delle repliche disponibili, ordinando lessico graficamente gli identificativi di tutte le repliche che hanno inviato il messaggio di *LIVE*.

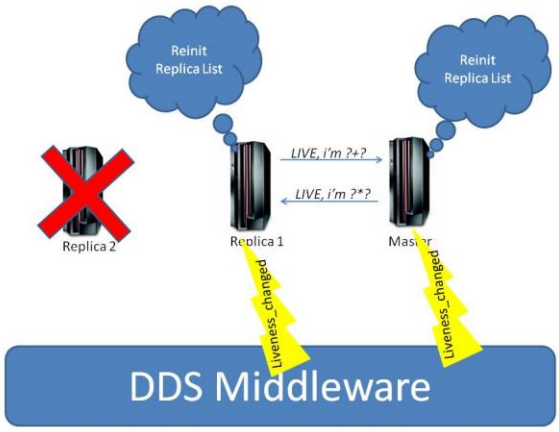


**Figura 7. Processo di riconfigurazione del sistema di Relay fronte della failure di una copia.**

Anche qui, grazie alla predisposizione di una coda temporanea delle news vecchie e alla presenza di un meccanismo di sincronizzazione tra la copia master e le varie repliche, in caso di *failure* del master sarà possibile evitare di perdere delle news, o quantomeno limitare il fenomeno. Sempre ovviamente che si effettui un giusto tuning dei parametri inerenti la massima lunghezza della coda dei messaggi vecchi e la frequenza della generazione delle notifiche di servizio da parte del master.

## V. Risultati dei test

### A. *Deploy*

I vari test sono stati effettuati nei vari laboratori nessi a disposizione della facoltà di Ingegneria, utilizzanzo mediamente una decina di macchine del laboratorio in aggiunta a due computer portatili di nostra proprietà. Le macchine dei laboratori utilizzate sono tutte macchine *single core*, tipicamente equipaggiate con processori *intel pentium 4* e banchi di *ram ddr* da 1GB, schede di rete *ethernet* di tipologia *baseT100* con velocità nominale di 100Mbps, mentre i due computer portatili sono equipaggiati con processori *dual core intel core 2 due T7400*, e banchi di ram in configurazione *dual channel* da 2G, mentre le scede di rete sono di tipo *baseT1000* con velocità nominale di 1Gbps.

Tutte le macchine dei laboratori utilizzate sono stati testati utilizzando il sistema operativo *windows xp*, mentre i due portatili hanno utilizzato rispettivamente *windows vista SP1* ed *Ubuntu 10.*

L'intero sistema è stato realizzato in linguaggio java.

La versione dell'implementazione DDS della Real-Time Innovations utilizzata è la 4.4.c (attualmente la più aggiornata), benché il sistema sia risultato correttamente funzionante anche utilizzando la versione 4.1.e per la quale è però necessaria una rigenerazione delle classi di supporto a partire dalle definizione dei dati contenute nei vari file *idl*.

### B. *Risultati*

Per quanto concerne i test riguardanti la replicazione è stato fatta un'analisi sui tempi medi di join di una nuova replica e i tempi medi di reazione del sistema a fronte della "caduta" di un nodo.

Per quanto riguarda il primo test, i dati forniti dai test sono risultati altalenanti e controversi, il grafico in Figura 8 indica l'andamento dei tempi di join al susseguirsi delle prove, il fenomeno strano rilevato, e che il grafico non mostra, e che il pattern qui indicato si ripete in maniera pressoché identica inesorabilmente ad ogni giornata di test. In pratica, costantemente dopo il primo tentativo della giornata la cui durata è significativamente superiore a quella dei seguenti, il tempo di join si assesta su valori intorno al secondo e mezzo. Questo fenomeno, all'apparenza anomalo, può essere spiegato con il fatto il middlware DDS, utilizzato, nel corso del primo tentativo di join debba effettuare il *discovery* sulla rete dei vari subscriber dei messaggi di controllo, come il join, senza avere alcuna informazione sui nodi DDS eventualmente attivi sulla rete. Mano a mano che il processo di discovery procede e rileva dei nuovi nodi DDS, la componente locale del middleware di rete mantiene in memoria volatile le informazioni inerenti gli indirizzi di rete dei nodi rilevati, così da poterli utilizzare in seguito come candidati primari per i successivi processi di discovery. Di qui, la drastica riduzione dei tempi di join nei successivi tentativi, che possono giovare delle informazioni rilevate precedentemente per effettuare un discovery mirato e più intelligente.

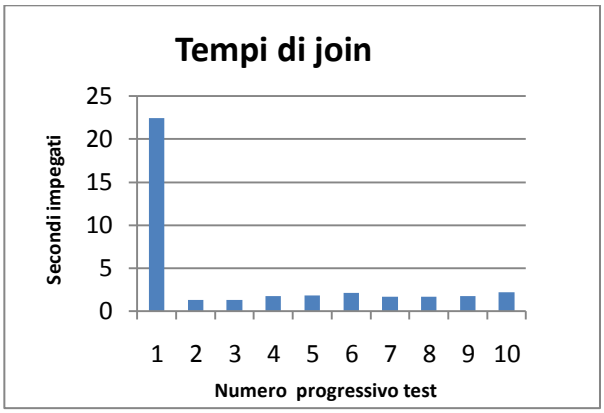


**Figura 8. Tempi di join di una nuova replica**

I tempi rilevati risultano essere pressoché identici sia nel caso del join di una replica del sistema di consegna che in quello del sistema di relay.

L'altro test effettuato è stato quello inerente i tempi di riconfigurazione del sistema a fronte di failure.

In particolare si è cercato di mettere in relazione i tempi medi di reazione del sistema ai tempi minimi e massimi teorici stimati per la rilevazione della failure (Figura 9).

Prendendo in considerazione i tempi di invio e rilevamento degli hearbeat è possibile identificare dei limiti teorici sui tempi minimi e massimi di reazione del sistema. Tuttavia non bisogna dimenticare che oltre

alla rilevazione dell'evento, la riconfigurazione del sistema implica la realizzazione di politiche di riallocazione delle risorse che richiedono sincronizzazione degli stati delle copie.

Per quanto riguarda il tempo minimo di rilevazione della failure siamo nel caso in cui l'ultimo heartbeat del publisher che "cade" giunge nel medesimo istante in cui la componente locale del middlware rileva la liveness, ed da quel momento in poi nessun heartbeat viene più generato dal publisher. Esattamente dopo 2400 millisecondi al subscriber del nodo locale viene notificato dal middleware l'evento di *liveness_change*.

Analogamente nel caso peggiore cui l'ultimo heartbeat del publisher che "cade" giunge nell'istante immediatamente successivo a quello in cui la componente locale del middlware rileva la liveness, ed da quel momento in poi nessun heartbeat viene più generato dal publisher. Alla successiva verifica da parte del middlware locale della liveness, il nodo caduto viene rilevato come ancora "vivo", a causa di quell'ultimo heartbeat e solo nell'intervallo di verifica successiva, dopo quindi 4800 millisecondi, sarà in grado di notificare al subscriber del nodo locale l'evento di *liveness_change*.

Durante i test i tempi di riconoscimento di una failure si sono attestati su un valore medio di 2900 ms.

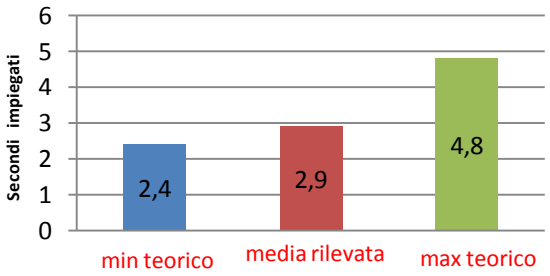


Figura 8. Tempi di join di una nuova replica

Anche in questo caso i tempi medi rilevati sia nel caso di failure di repliche del sistema di consegna di repliche del sistema di relay non presentano significative variazioni.

Tutti i test fino ad ora indicati fanno riferimento sia a situazioni di traffico intorno al 50% della banda teorica disponibile per la rete. Tuttavia questi valori non risultano essere particolarmente suscettibili a problematiche di traffico in quanto utilizzano delle policy di elevata qualità di servizio, offerte in maniera nativa dalle architetture basate sullo standard DDS, che prevedono la gestione ed inoltro prioritario di questo tipo di dati.

## VI. Conclusioni

La progettazione e la successiva realizzazione del sistema RE.VE.N.GE. ha rappresentato, per il nostro gruppo, una sfida, dovuta sia all'utilizzo di nuovi strumenti sia al modo in cui questi strumenti sono stati utilizzati.

Nelle primissime fasi si è reso necessario uno studio approfondito dell'architettura DDS ed in particolare dell'implementazione fornita da RTI. Il manuale ricchissimo di informazioni e corredato di esempi rappresenta una buona base di partenza.
Benchè la qualità di questa implementazione sia elevatissima, risulta particolarmente complesso effettuare il debug delle applicazioni che la utilizzano a causa di alcune eccezioni che spesso vengono sollevate senza alcun dettaglio riguardo ai fattori generanti delle stesse; anche se per fortuna è presente sul sito web del produttore una ricchissima raccolta di soluzioni a i problemi più comuni. Inoltre l'utilizzo "atipico" di questo strumento ha reso il tutto ancora più complesse.

Tuttavia una volta compresa la logica di funzionamento del middleware tutto diventa semplice e naturale, a dimostrazione di ciò la, relativa, ridotta quantità di codice scritta da parte nostra per realizzare l'intero sistema. La sconfinata quantità di policy permette la realizzazione delle più disparate modalità di interazione, permettendo così di concentrarsi sulla logica del programma più che sull'interazione.

Proprio grazie alla potenza dell'implementazione DDS di RTI ed alla sua flessibilità è stato possibile concentrare i nostri sforzi sulla definizione di un'architettura il più possibile scalabile ed affidabile, capace di realizzare un buon compromesso tra efficienza ed affidabilità del servizio offerto. In nostro sistema è in grado di gestire grandi moli di dati e utenti. Infatti, come sarà possibile rilevare dal lavoro di Luca Nardelli, l'utilizzo del sistema da noi implementato ha un costo computazionale e di memoria piuttosto contenuto; mentre il collo di bottiglia principale risulta essere la disponibilità di banda, problematica fisiologica e non semplicemente affrontabile. Proprio questo collo di bottiglia ci ha impedito di spingere il nostro sistema al proprio limite di funzionamento così da poter effettuare maggiori considerazioni in termini di performance.

Nel complesso siamo molto soddisfatti del lavoro svolto, sia per i risultati ottenuti, che per le nuove capacità e conoscenze acquisite.