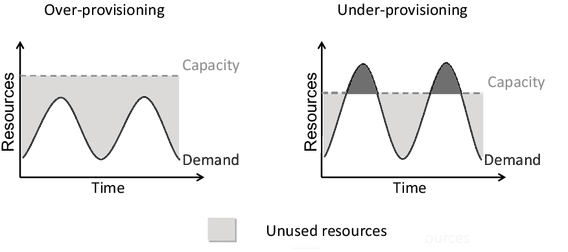
SDCC Progetto A1 - JDSys

Sistema di Storage Distribuito di tipo Chiave Valore

Danilo Dell’Orco  
 Facoltà di Ingegneria  
 Università di Roma Tor Vergata  
 Roma, Lazio, Italia  
 danilodellorcolp@gmail.com

Jacopo Fabi  
 Facoltà di Ingegneria  
 Università di Roma Tor Vergata  
 Roma, Lazio, Italia  
 jacopo.fabi.1997@gmail.com

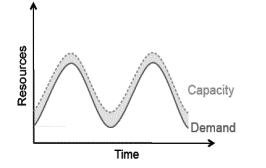


ABSTRACT

JDSys è un sistema di storage distribuito di tipo Chiave valore. Il sistema è distribuito nel senso che i dati salvati sono distribuiti su molteplici server, fornendo quindi un servizio ad alta disponibilità ed affidabilità. JDSys mira a funzionare su un'infrastruttura composta da diversi nodi, il cui numero preciso viene selezionato in modo da avere una distribuzione equa del carico di lavoro. Per questo il nostro sistema utilizza i servizi AWS, in modo da avere a disposizione una scalabilità teoricamente infinita. JDSys fornisce anche un’integrazione con il cloud storage S3 per aumentare anche la scalabilità al numero di dati. In particolare vengono migrate sul cloud tutte le entry scarsamente accedute, in modo tale da liberare spazio per entry più utilizzate. Oltre ad essere distribuiti, i dati vengono anche replicati tra i diversi server; questo garantisce una migliore scalabilità al numero di utenti e di dati.

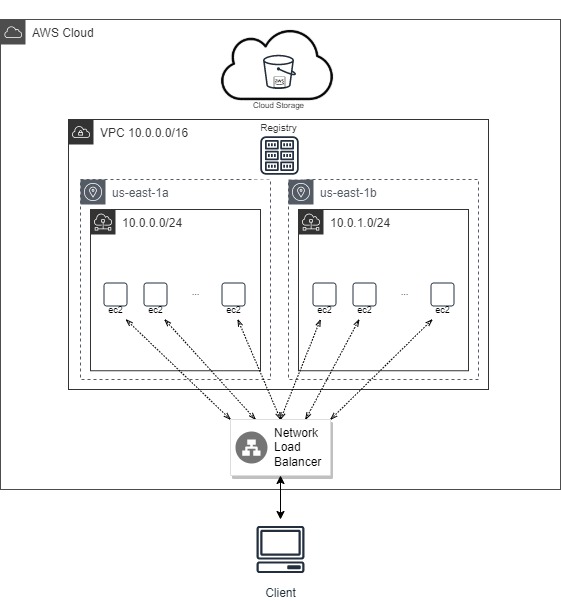
1. INTRODUZIONE

Nel corso degli anni il traffico internet è cresciuto a dismisura. Ciò significa che un qualsiasi sistema che riceve richieste tramite la rete deve gestire una quantità di dati e di operazioni sempre maggiori. In un simile scenario è importante mantenere lo stesso livello di prestazioni ed affidabilità, a prescindere dal carico di lavoro in ingresso. Gli approcci tradizionali consistono nello di scalare verticalmente l’hardware dei server. Tuttavia questo approccio ha un costo esponenziale, ed ha quindi un upper bound al livello di prestazioni raggiungibili. Un altro problema da affrontare è quello del dimensionamento della propria infrastruttura, ovvero quante risorse allocare per fornire il servizio. In un approccio classico si potranno allocare soltanto un numero fisso di risorse durante l’esecuzione del servizio. Ciò vuol dire che, a fronte di un numero di richieste variabile, si potranno avere due scenari. Nell’approccio di under provisioning si allocano un numero di risorse che non riescono a far fronte a picchi di carico. Quindi si gestisce normalmente un carico medio, ma durante I picchi il servizio non sarà disponibilile. Nell’ over provisioning invece si forniscono un numero grande di risorse, in modo da poter gestire anche i picchi di carico. Questo garantisce che il sistema sia sempre disponibile, ma ci sarà anche un numero importante di sprechi in quanto bisognerà avere un'infrastruttura robusta anche durante il traffico medio.

Sfruttando invece un’infrastruttura cloud, è possibile offrire un servizio elastico, che permette di allocare risorse in base alla quantità di traffico che bisogna gestire. Questo approccio permette di minimizzare i costi, offrendo al contempo un servizio sempre disponibile.

L’altro vantaggio offerto dall’utilizzo dell’infrastruttura cloud è quello della scalabilità, poiché si può scalare teoricamente fino ad un numero infinito di macchine sfruttando la scalabilità orizzontale anziché quella verticale.

JDSys è un servizio di storage distribuito di tipo chiave-valore. Questo può ricevere richieste concorrenti, ed un traffico variabile. Per i motivi precedentemente descritti, JDSys utilizza Amazon Web Services a diversi livelli. Le istanze EC2 hostano i singoli data server, il Load Balancer garantisce una suddivisione equa delle richieste, mentre L’Auto Scaling permette di avere sempre il corretto numero di risorse attive in base al traffico in ingresso al sistema. Per garantire una ricerca efficiente della chiave all’interno del sistema, i diversi server sono organizzati in una overlay network ad anello (chord). Inoltre, per avere una maggirore scalabilità e disponibilità, i nodi implementano delle tecniche per la replicazione e riconciliazione dei dati.

Questo documento è strutturato come segue. La sezione 2 descrive il modello di dati utilizzato. La sezione 3 presenta la panoramica delle API client. La sezione 4 presenta la struttura e la progettazione del sistema. La sezione 5 descrive nel dettaglio i principali aspetti implementativi. La sezione 6 descrive i test effettuati su JDSys. La sezione 7 conclude il documento, con le considerazioni finali sul sistema.

2. COMPONENTI

JDSys basa lo storage sul concetto di **entry**. Una entry è coppia chiave-valore. La Chiave rappresenta un identificativo univoco per l’entry, mentre possono esserci anche stessi valori associati a chiavi diverse. La chiave è rappresentata tramite una stringa, mentre il valore è un array di stringhe in quanto può contenere un numero variabile di argomenti.

Il sistema di storage è distribuito su diversi server denominati Nodi. I nodi sono strutturati in una overlay network ad anello, chiamato JDSys Ring.

Il Service Registry è un componente centralizzato che permette la scoperta reciproca tra i nodi. Questo comunica con il load balancer, e periodicamente aggiorna la lista dei nodi healthy, che quindi faranno parte del JDSys Ring.

3. CLIENT API

JDSys permette all’utente di aggiungere, rimuovere, leggere o modificare le entry tramite quattro operazioni: get(), put(), delete() e append():

* *get(key)* ricerca la chiave all’interno del sistema e, se questa esiste, restituisce l’entry ad essa associata.
* *put(key,value)* inserisce l’ entry specificata tramite chiave-valore all’interno del sistema. Se la chiave già esiste, l’entry associata verrà aggiornata con il nuovo valore specificato.
* *delete(key)* elimina dal sistema l’ entry associata alla key specificata
* *append(key,arg1)* modifica l’entry specificata tramite la chiave key, aggiungendo al suo valore l’argomento arg1

4. ARCHITETTURA DI SISTEMA

Un sistema di storage distribuito, oltre alla persistenza dei dati, deve offrire anche soluzioni scalabili e robuste per il bilanciamento del carico, il provisioning delle risorse, la sincronizzazione delle repliche, la gestione del sovraccarico, il trasferimento dello stato e la concorrenza, l'assegnazione delle richieste e l'instradamento delle richieste, il monitoraggio e gli allarmi del sistema. JDSys, fornisce una soluzione ad ognuno di questi aspetti, ed in questa sezione andiamo quindi a descrive nel dettaglio come sono stati affrontati.

Parleremo quindi in particolare della topologia di rete utilizzata, dei meccanismi di comunicazione, dei protocolli utilizzati, e per ognuno di questi punti verranno quindi descritti i principali aspetti implementativi.

4.1 Infrastruttura di Rete

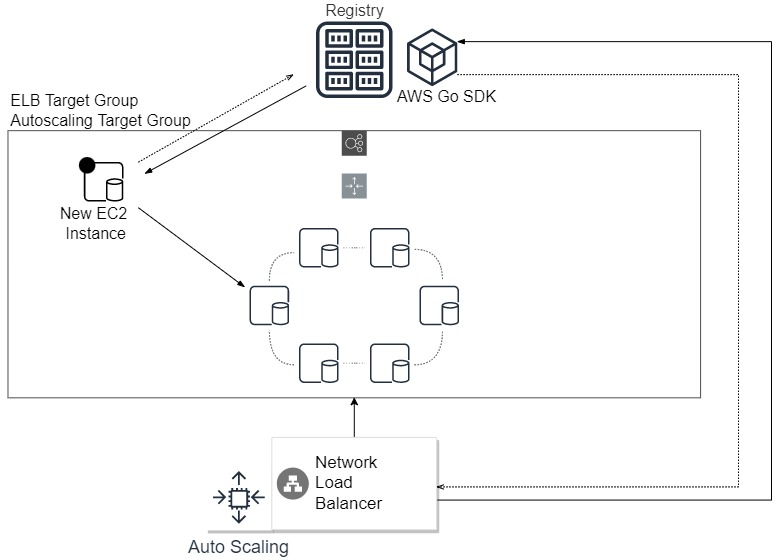
L’intera infrastruttura di rete è stata definita sfruttando Amazon Web Service. Ciò ha permesso di avere un sistema distribuito I cui nodi dell’applicazione sono hostati sul cloud.

La Figura 1 mostra l’infrastruttura di rete di JDSys. I nodi del sistema sono hostati su delle istanze Amazon EC2 micro, che forniscono delle limitate capacità computazionali e di storage.

Risulta fondamentale in questo senso distribuire equamente il carico di lavoro tra questi nodi. A tale scopo, si utilizza l’ Elastic Load Balancer di AWS, che permette di distribuire le richieste in ingresso verso le istanze registrate nel Target Group. Un client dovrà quindi contattare il load balancer tramite il suo URL DNS, e la sua richiesta verrà redirezionata in automatico su uno dei nodi del Ring JDSys. Figura 1: Infrastruttura di Rete **JDSys**

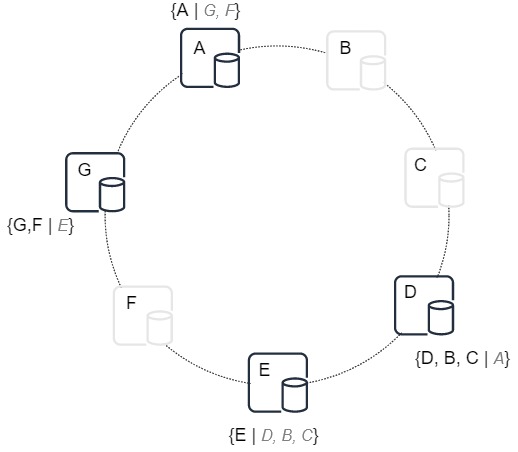
Per motivi di sicurezza ed efficienza, tutte le istanze EC2 utilizzate sono state istanziate all’interno di un Virtual Private Cloud. In questo modo i nodi possono essere contattati dal Load Balancer, e possono comunicare tra loro tramite indirizzi IP privati, ma non potranno essere contattati direttamente dall’esterno.

Per simulare in maniera più fedele una latenza di rete, i nodi vengono istanziati in due zone di disponibilità differenti, che descrivono quindi due sottoreti IP differenti (10.0.0.0/24 e 10.0.0.1/24). Il service registry si trova anch’esso all’interno della VPC. A questa istanza è stato assegnato un IP Elastico, in modo che tutti i nodi possano contattarlo con successo.

Fuori dal VPC troviamo invece un Bucket S3, che fornisce lo storage Cloud necessario per una maggiore scalabilità ai dati. I nodi interagiscono con S3 utilizzando le SDK di AWS, sfruttando quindi la rete internet pubblica in questo caso.

4.2 Autoscaling & Cloudwatch

Un aspetto fondamentale di JDSys è quello della scalabilità automatica. E’ stato utilizzato a tale scopo il sistema di Autoscaling fornito da AWS, che permette di definire un numero minimo e massimo di istanze da istanziare in modalità on-premise. In JDSys sono state utilizzate un minimo di 3 istanze attive, fino ad un massimo di 10. Si noti come il parametro massimo è impostato ad un valore “basso” in virtù delle limitazioni fornite da AWS Educate.

Per monitorare il traffico di richieste verso il sistema, si utilizza il servizio Cloudwatch. La metrica utilizzata è il numero di pacchetti in ingresso al gruppo di autoscaling, in quanto fornisce un indice indicativo sul numero di richieste e quindi anche sul numero di operazioni che ogni nodo dovrà gestire complessivamente.

Analizzando i dati forniti da questo servizio, sono stati ricavati i valori critici in termini di traffico. Si è osservato infatti che per un traffico medio di 30000 pacchetti, i nodi possono andare in saturazione (out of memory) non riuscendo più a gestire un numero elevato di richieste concorrenti. Analogamente si è osservato che Per traffico medio sotto le 8000 richieste, il sistema garantisce sempre la normale operatività.

Sono stati quindi definiti due allarmi, uno per lo Scale In ed uno per lo Scale Out, proprio sfruttando i dati ricavati in precedenza. Lo Scale Out consiste nella rimozione di un’istanza dal gruppo di autoscaling, e viene attivato quando il traffico medio in ingresso risulta superiore a 10000 pacchetti per più di 5 minuti. Si è deciso di mantenere questa soglia pari ad 1/3 di quella critica in modo da fornire al servizio di autoscaling il tempo necessario ad istanziare la macchina e a rendere UP & Running il JDSys Node.

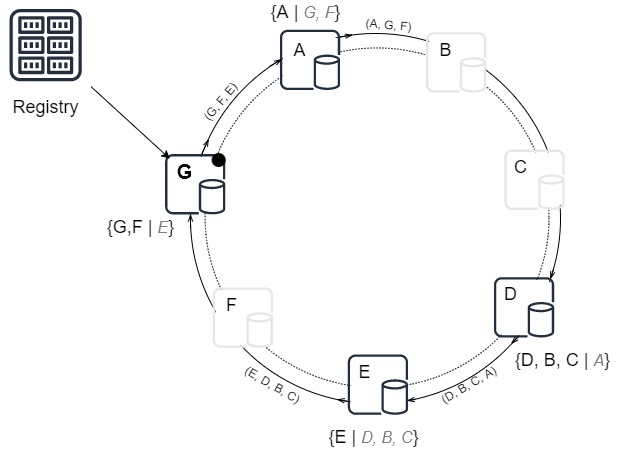
4.3 Overlay Network & Partitioning

Uno dei requisiti di progettazione chiave per JDSys è che deve scalare proporzionalmente al numero di richieste in ingresso al sistema. Ciò richiede un meccanismo per partizionare dinamicamente i dati sull'insieme di nodi, ed avere al contempo un metodo efficace di lookup su queste risorse distribuite.

A tale scopo, JDSys utilizza l’algoritmo Chord. Ai nodi e alle chiavi viene assegnato un identificatore utilizzando il Consistent Hashing. Utilizzando il protocollo di lookup di Chord, i nodi e le chiavi sono disposti in una struttura ad anello, in cui ogni nodo ha un successore e un predecessore. Il successore di un nodo è il nodo successivo nel cerchio identificativo in senso orario. Il predecessore è in senso antiorario. Il concetto di successore viene utilizzato anche per le identificare il nodo che deve gestire una determinata entry. Il nodo successore di una chiave k è il primo nodo il cui ID è uguale a k o segue k all’interno dell’anello. Ogni chiave è assegnata al suo nodo successore, per cui la ricerca di una chiave consiste nel fare una richiesta verso succ(k).

Per evitare la ricerca lineare sopra, Chord implementa un metodo di ricerca più veloce richiedendo a ciascun nodo di mantenere una finger table. Per il nodo n, l'entry i-esima di questa tabella mantiene l'indirizzo IP del nodo n+2^{i-1})mod2^m. Ogni volta che un nodo vuole cercare una chiave k, passerà la query al successore o predecessore più vicino alla chiave k, basandosi sulla sua Finger Table. L'idea è quindi quella per cui un nodo conosca bene i nodi nelle sue vicinanze, ed abbia solo una conoscenza approssimativa dei nodi più lontani. Questo algoritmo di lookup, rende la ricerca estremamente più efficiente, in quanto il numero di nodi che devono essere contattati per trovare un successore in una rete di N nodi è O(log(N))

Un altro vantaggio offerto da questo algoritmo, è che l’ingresso (leave) o l'arrivo (join) di un nodo, modifica solo le finger table dai suoi vicini immediati mentre gli altri nodi rimangono inalterati. Questo risulta fondamentale nell’ architettura elastica di JDSys in cui le risorse vengono allocate e deallocate dinamicamentesdf

Un nodo, all’ingresso nel sistema, contatterà il service registry che gli fornirà una lista di istanze attive. Il nodo contatterà quindi un’istanza casuale per inserirsi nell’anello ed aggiorna le sue informazioni su successore e predecessore. Per inizializzare il suo storage locale, il nuovo nodo chiederà al predecessore di inviargli le sue entry, diventando quindi il gestore delle chiavi secondo il Consistent Hashing.

Al leave, invece il nodo contatterà il suo successore informandolo della terminazione. A questo invierà tutte le sue entry e conseguentemente verranno aggiornate tutte le informazioni delle finger table

4.4 Replicazione

Per ottenere **una maggiore** disponibilità e **scalabilità**, **JDSys** replica i propri dati su più **nodi**. Ogni nodo JDSys sarà responsabile di un certo sottoinsieme di entry, ed allo stesso tempo si occuperà di replicare tali entry sul suo nodo successore.

Il successore secondo chord di una chiave k svolge il ruolo di gestore di tale chiave. Il gestore è responsabile quindi, oltre che della memorizzazione di tali dati, anche della loro replicazione sul nodo successivo. Nella Figura ??, il nodo A replica la chiave A di cui è gestore sul nodo D. Il nodo D replica le chiavi B,C,D sul nodo E. Il nodo E replica la chiave E di cui è gestore sul nodo G. Il nodo G replica le chiavi G ed F sul nodo A.

La replicazione avviene non appena si riceve una nuova entry o un aggiornamento su di essa. Viene immediatamente contattato il nodo successore inviandogli la replica.

Sfruttando questo meccanismo, migliora inoltre la consistenza ai guasti, in quanto al crash di un nodo, le sue informazioni non andranno perse ma saranno presenti su almeno un altro nodo. Viene quindi demandata a Chord la ricostruzione dell’anello e quindi la riorganizzazione dei gestori delle varie chiavi.

4.5 Consistenza

JDSys offre un modello di consistenza finale; ciò vuol dire che per alcuni lassi di tempo le repliche su diversi nodi possono essere inconsistenti tra loro, e la consistenza verrà raggiunta soltanto in un secondo momento. Quando un nodo riceve una richiesta di scrittura o aggiornamento, risponderà immediatamente al chiamante, e parallelamente procederà ad inviare la replica (o aggiornarla) presso il suo successore. Una chiamata put() può quindi fornire la risposta al suo chiamante prima ancora che l'aggiornamento sia stato effettuato ed applicato a tutte le repliche; ciò vuol dire che possono verificarsi scenari in cui un'operazione di get() può restituire un oggetto che non dispone degli aggiornamenti più recenti, in quanto l’aggiornamento deve ancora essere propagato.

*4.4.1 Riconciliazione*

Per risolvere i conflitti tra tutte le repliche, giungere così alla consistenza finale, si utilizza un algoritmo di riconciliazione di tipo Asyncrhonous Repair. Periodicamente il service registry avvia il servizio di riconciliazione, contattando un nodo a caso tra quelli dell’anello. Questo nodo sarà il gestore della riconciliazione, ed invierà le proprie entry al nodo successivo. Il nodo che riceve questo messaggio, utilizzerà l’aggiornamento ricevuto per riconciliare le sue repliche, e successivamente invierà a propria volta le sue entry al suo successore.

Per risolvere un conflitto tra due entry con la stessa chiave, si utilizza la tecnica Last Write Wins, mantenendo quindi la copia più recentemente aggiornata. In tal senso, ad ogni scrittura su una entry si aggiorna il suo timestamp, sfruttando il Network Time Protocol che garantisce un’accuratezza di 100ms.

Questo processo di riconciliazione, per avere la certezza di risolvere tutte le copie conflittuali, necessita al più di due giri dell’anello. Questo perché può verificarsi lo scenario in cui, per una chiave k, la copia aggiornata sia in possesso del predecessore del gestore della replicazione, e quindi con un solo giro nessun nodo riconcilierà la sua copia di k.

Per quanto riguarda invece le operazioni di delete(), si è utilizzata una semantica un po' più forte. Questo perché con il meccanismo di replicazione, può verificarsi lo scenario in cui delle copie eliminate “tornino in vita”. Per evitare questo, al momento dell’eliminazione di un’entry, viene immediatamente propagata la richiesta su tutto l’anello e non solo verso il successore. Questo perché a seguito del join o del leave, i nodi possono avere delle repliche fuori dalla portata del gestore. Se queste verranno comunque aggiornate grazie alla riconciliazione, ciò non sarà vero in caso di cancellazione.

4.6 Cloud Storage

Per migliorare la scalabilità ai dati, JDSys integra i propriservizi con un Bucket S3. Questo bucket è utilizzato globalmente da tutti i nodi JDSys per migrare le entry che sono scarsamente accedute. Tutte le entry non accedute da almeno 30 minuti (sia in lettura che in scrittura) verranno rimosse dallo storage locale e verranno caricate sullo storage S3. Ogni nodo mantiene quindi traccia delle entry presenti sul cloud, e alla prima richiesta ricevuta per una di quelle entry, procederà a scaricarla nuovamente da S3 ed inserirla nel suo storage locale.

5. IMPLEMENTAZIONE

JDSys è sviluppato utilizzato il linguaggio Go. Il vantaggio principale è dovuto alla gestione efficiente della concorrenza grazie alle goroutine ed ai canali. Inoltre permette facilmente di utilizzare le Remote Procedure Call, che semplificano di molto l’interazione client-server o nodo-nodo.

In JDSys, ogni nodo ha quattro componenti software principali: listener delle richieste, rilevamento di appartenenza e guasti e un motore di persistenza locale. Tutti questi componenti sono implementati in Java. Il componente di persistenza locale di Dynamo consente di collegare diversi motori di archiviazione. I motori in uso sono Berkeley Database (BDB) Transactional Data Store2, BDB Java Edition, MySQL e un buffer in memoria con archivio di supporto persistente. Il motivo principale per progettare un componente di persistenza collegabile è scegliere il motore di archiviazione più adatto ai modelli di accesso di un'applicazione. Ad esempio, BDB può gestire oggetti tipicamente nell'ordine di decine di kilobyte mentre MySQL può gestire oggetti di dimensioni maggiori. Le applicazioni scelgono il motore di persistenza locale di Dynamo in base alla distribuzione delle dimensioni degli oggetti. La maggior parte delle istanze di produzione di Dynamo utilizza BDB Transactional Data Store. Il componente di coordinamento delle richieste è costruito su un substrato di messaggistica guidato dagli eventi in cui la pipeline di elaborazione dei messaggi è suddivisa in più fasi simili all'architettura SEDA [24]. Tutte le comunicazioni sono implementate utilizzando i canali Java NIO. Il coordinatore esegue le richieste di lettura e scrittura per conto dei client raccogliendo i dati da uno o più nodi (nel caso delle letture) o memorizzando i dati su uno o più nodi (per le scritture). Ogni richiesta del client determina la creazione di una macchina a stati sul nodo che ha ricevuto la richiesta del client. La macchina a stati contiene tutta la logica per identificare i nodi responsabili di una chiave, inviare le richieste, attendere risposte, eseguire eventuali tentativi, elaborare le risposte e impacchettare la risposta al client. Ogni istanza della macchina a stati gestisce esattamente una richiesta client. Ad esempio, un'operazione di lettura implementa la seguente macchina a stati: (i) inviare richieste di lettura ai nodi, (ii) attendere il numero minimo di risposte richieste, (iii) se sono state ricevute poche risposte entro un determinato tempo, fallisce il richiesta, (iv) altrimenti raccogliere tutte le versioni dei dati e determinare quelle da restituire e (v) se il versioning è abilitato, eseguire la riconciliazione sintattica e generare un contesto di scrittura opaco che contiene il clock vettoriale che sussume tutte le versioni rimanenti. Per ragioni di brevità, gli stati di gestione degli errori e tentativi sono stati omessi.

KEYWORDS

Insert keyword text, Insert keyword text, Insert keyword text, Insert keyword text

JDSys organizza i nodi secondo una struttura ad anello. Una qualsiasi richiesta di lettura/scrittura per una chiave viene instradata a qualsiasi nodo nel cluster Cassandra. Il nodo determina quindi le repliche per questa particolare chiave. Per le scritture, il sistema instrada le richieste alle repliche e attende che un quorum di repliche riconosca il completamento delle scritture. Per le letture, in base alle garanzie di coerenza richieste dal client, il sistema o indirizza le richieste alla replica più vicina oppure indirizza le richieste a tutte le repliche e attende un quorum di risposte.

ACM Reference format:

FirstName Surname, FirstName Surname and FirstName Surname. 2018. Insert Your Title Here: Insert Subtitle Here. In *Proceedings of ACM Woodstock conference (WOODSTOCK’18). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1 Insert Heading Level 1

The updated template, user manuals, samples, and required fonts, all are available at the URL <https://www.acm.org/publications/proceedings-template>. It contains said information for all three versions of MS Word (Windows and 2 versions of Mac). There are also separate links to the user guide, which can be referred to by the user. This URL also contains some useful video links, which describe how to add the template, structure the paper, and generate the layout, in different clips. **Display Formula with Number**

 (1)

**Continuation part of Paragraph Text** The user must style this paragraph in **ParaContinue** style, which follows immediately after the **DisplayFormula** (numbered equation). The **DisplayFormula** style is applied only in case of a numbered equation. A numbered equation always has a number to its right. Insert paragraph text here. **Display Formula without Number**



The **DisplayFormulaUnnum** style is applied only in case of an unnumbered equation. An unnumbered display equation never contains an equation number to its right, and this unique property distinguishes it from a numbered equation.



Figure 1: Figure Caption and Image above the caption [In draft mode, Image will not appear on the screen]

**Theorem/Proof/Lemma.** Insert text here for the enunciation or Math statement. Insert text here for the enunciation or Math statement. Insert text here for the enunciation or Math statement. Insert text here for the enunciation or Math statement. Insert text here for the enunciation or Math statement.

....Insert text here for the Quotation or Extract, Insert text here for the Quotation or Extract, Insert text here for the Quotation or Extract, Insert text here for the Quotation or Extract, Insert text here for the Quotation or Extract, Insert text here for the Quotation or Extract.

1.1 Heading Level 2

In the below paragraph, it is explained how alt-txt value is placed in **MS Word 2010**. To add alternative text to a picture in Word 2010, follow these steps:

1. In a Word 2010 document, insert a picture.
2. Right click on the inserted picture and select the **Format Picture** option.
3. Select the **Alt Txt** option from the left-side panel options.
4. In the "Title:" and "Description:" text boxes, type the text you want to represent the picture, and then click "Close".

Below are steps to place alt-txt value in **MS Word 2013/2016**. To add alternative text to a picture in Word 2013/2016, follow these steps:

1. In a Word 2013/2016 document, insert a picture.
2. Right click on the inserted picture and select the **Format Picture** option.
3. In the settings at the right side of the window, click on the "Layout & Properties" icon (3rd option).
4. Expand **Alt Txt** option.
5. In the "Title:" and "Description:" text boxes, type the text you want to represent the picture, and then click "Close".

*1.1.1 Heading Level 3.* Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here.

*1.1.1.1 Heading Level 4.*Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here.

ACKNOWLEDGMENTS

Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here.

REFERENCES

[1] Patricia S. Abril and Robert Plant, 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan, 2007), 36-44. DOI: <https://doi.org/>10.1145/1188913.1188915.

[2] Sten Andler. 1979. Predicate path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '79)*. ACM Press, New York, NY, 226-236. DOI:https://doi.org/10.1145/567752.567774

[3] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. DOI:https://doi.org/10.1007/3-540-09237-4.

[4] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY..

Conference Name:ACM Woodstock conference

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00