**Università degli Studi di Pisa**

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in Informatica


Tesi di laurea

# Rilevazione Automatica di Disservizi di Rete


Candidato:
**Lorenzo Brunetti**
**Matricola 451165**

Relatore:
**Luca Deri**



**Anno Accademico 2014-2015**

Un grazie speciale
Alla mia famiglia
A Cristina
per aver supportato
e sopportato le mie scelte

# Indice

1 Introduzione 3

2 Motivazioni 5

3 Requisiti 8
- 3.1 Requisiti Funzionali . . . . . . . . 8
  - 3.1.1 Creare uno Storico delle Interrogazioni . . . . . . . . 8
  - 3.1.2 Ottimizzare il Numero delle Richieste . . . . . . . . 8
  - 3.1.3 Variare i Tempi di Interrogazione . . . . . . . . 9
  - 3.1.4 Ottenere un Sistema Aperto . . . . . . . . 9
  - 3.1.5 Correlare Metriche delle Interrogazioni . . . . . . . . 10
  - 3.1.6 Valutare l' Andamento . . . . . . . . 10
- 3.2 Requisiti non Funzionali . . . . . . . . 11
  - 3.2.1 Ottenere la Massima Robustezza . . . . . . . . 11
  - 3.2.2 Massimizzare la concorrenza . . . . . . . . 11
  - 3.2.3 Rendere il Sistema Facilmente Configurabile . . . . . . . . 12
  - 3.2.4 Integrabile con Sistemi Esistenti . . . . . . . . 12

4 Stato dell' Arte 14
- 4.1 Rilevare Guasti in modo Passivo . . . . . . . . 14
- 4.2 Le Tecniche di Rilevazione Attiva . . . . . . . . 15
- 4.3 Soluzioni per il Monitoring Presenti sul Mercato . . . . . . . . 16
  - 4.3.1 Zabbix . . . . . . . . 16
  - 4.3.2 Nagios . . . . . . . . 17
- 4.4 Il Monitoring Come Servizio Cloud . . . . . . . . 19
  - 4.4.1 Datadog . . . . . . . . 20
  - 4.4.2 TrueSight Pulse . . . . . . . . 21
- 4.5 Uno Sguardo Riassuntivo sulle Soluzioni Presenti . . . . . . . . 21

5 Architettura della Soluzione 23
- 5.1 Struttura Architetturale . . . . . . . . 23
- 5.2 Il Filtraggio dei Dati Attraverso i Selettori . . . . . . . . 31
- 5.3 Robustezza della soluzione . . . . . . . . 34

6 Implementazione e Validazione 36
  - 6.0.1 Riferimenti al Codice Sorgente . . . . . . . . 39
- 6.1 Validazione attraverso un Single-board computer . . . . . . . . 39
- 6.2 Considerazioni sulle Prestazioni . . . . . . . . 41
- 6.3 Problematiche Aperte . . . . . . . . 43

7 Conclusioni 44

# 1 Introduzione

Il mondo delle reti di calcolatori sta assumendo dimensione sempre più estesa ed eterogenea.
Il loro ruolo diventa quindi sempre più fondamentale per la miriade di impieghi per cui vengono sfruttate.
Negli ultimi anni le reti hanno subito una forte estensione a causa dell'aumentare esponenziale dei dispositivi ad essa connessi. Si assiste ad una varietà diversa di dispositivi come smartphone, notebook, tablet che necessitano di accedere a una varietà di servizi in costante aumento.

Con la crescita esponenziale del carico sulle reti, si rende fondamentale garantire il corretto e costante funzionamento di un numero enorme di servizi anche in condizioni di stress o anomalia.
In questo scenario, è intuibile come il monitoring assuma un ruolo cruciale per le reti odierne. Con monitoring vogliamo rappresentare una vasta area composta di software il cui principale compito è quello di evidenziare le situazioni anomale che si presentano in un determinato sistema.
In un contesto eterogeneo come quello attuale, i software di monitoring necessitano di un livello di astrazione molto elevato; il concetto di rete deve essere quindi generalizzato il più possibile.
Nel nostro percorso si analizzeranno prevalentemente reti di calcolatori ma di tanto in tanto faremo riferimenti a contesti che esulano da quello classico in modo da marcare come i concetti del monitoring siano estensibili anche a contesti non strettamente connessi alle reti.
Il concetto di rete potrebbe essere astratto come un insieme di nodi in grado di comunicare informazioni circa la loro condizione, tenendo bene in mente questa definizione, si sarà costretti a definire concetti e architetture, il più trasversali possibile.
Dopo aver posto questo vincolo sull'impostazione da seguire, si troveranno una serie di necessità insoddisfatte. Si focalizzerà l'attenzione sulla mancanza di un sistema per il mantenimento dei dati rilevati, che, in molti casi pratici, rappresenta un ostacolo al fine di conoscere l'evoluzione di una certa componente. Ciò pone in una posizione di svantaggio nel comprendere lentità di un guasto.

Oltre a ciò si porrà il problema di come valorizzare i dati di un ipotetico storico in modo da renderlo uno strumento utile e non solo un catalogo di eventi fine a se stesso. Si analizzeranno queste situazioni e le riflessioni da esse derivate nella sezione motivazioni.
Si stileranno poi, una serie di punti cardine fondamentali per delineare gli scopi della tesi.
Tra questi rientreranno per esempio: la capacità di effettuare controlli più dettagliati su di un certo risultato o il gestire in maniera efficiente le richieste

verso una certa componente.
Questo tipo di considerazioni sarà parte portante della sezione requisiti la quale raccoglierà in dettaglio tutte le ragioni che guideranno il percorso all' interno della sezione stato dell' arte.
Con una serie di obiettivi ben definiti nella sezione dei requisiti, si andrà a scoprire quello che è già disponibile ad oggi, con il fine di capire quali degli obiettivi stilati, sono in realtà già risolti da soluzioni esistenti.
Per fare ciò si partirà da quello che la letteratura scientifica offre, si osserveranno in dettaglio due approcci, quello del monitoring passivo e il suo opposto, ovvero il monitoring attivo.
Per ognuno di questi si descriveranno quali sono le caratteristiche, i limiti ed i vantaggi che introducono, per poi analizzare alcune implementazioni commerciali, confrontando le funzionalità offerte con i requisiti stilati.
Alla fine della sezione stato dell'arte si capirà quali lacune e funzionalità un software di monitoring dovrà implementare al fine di coprire i requisiti imposti.
Si svilupperanno in dettaglio una serie di definizioni e concetti necessarie al raggiungimento degli scopi. Si descriveranno da dove queste prendono radice e come risolvano le mancanze rilevate nello stato dell'arte. Tali considerazioni saranno parte della sezione architettura della soluzione. Capendo quando sia fondamentale avere degli esempi pratici per capire la bontà delle scelte architetturali, si descriverà un caso d'uso in cui il software verrà sviluppato ed eseguito. Il contesto in cui si validerà sarà volto a creare quelle che definiremo "situazioni critiche",cioè ambienti in cui i requisiti imposti saranno portati all' estremo.
In tutti gli ambiti della tesi ma soprattutto in questa parte si cercherà di osservare il monitoring in modo non strettamente legato all' ambito di rete.
Si affronteranno questi aspetti nella implementazione e validazione al fine di capire quanto i requisiti saranno stati raggiunti e se questi sono validi sotto il profilo prestazionale.
Si riassumeranno infine, i risultati della tesi, compromessi fatti e sviluppi futuri in cui l' architettura potrà evolversi.

## 2 Motivazioni

Durante il percorso di studi più volte si è incontrato il problema di controllare il comportamento di una risorsa.
A seguito di alcuni corsi portati avanti durante il percorso di studi, ha suscitato particolare interesse il monitoraggio nel campo delle reti. Dopo aver appreso concetti e metodologie che ad oggi caratterizzano tale mondo, l'attenzione si è focalizzata sullo studio delle soluzioni software presenti in commercio.
L'uso di tali software ha portato definito una serie di motivazioni su quali siano i punti importanti e quali quelli da sviluppare.

In primo luogo, nell' esperienza maturata con i software di monitoring assume fondamentale rilevanza capire l'evoluzione che ha portato ad un guasto. Solitamente infatti una volta rilevata una condizione di fault, si hanno informazioni limitate su come l'intero sistema abbia evoluto il suo comportamento. Siamo in grado di visualizzare lo stato finale di una componente e di alcune ad essa correlate. Non si hanno quindi gli strumenti necessari per analizzare in modo completo un guasto.

Altro punto interessante è senza dubbio rappresentato dall' impatto del monitoring ha sulle prestazioni della rete.
Che si tratti di richieste passive o attive queste, in ogni caso, la loro esecuzione non è gratuita. Le componenti monitorate infatti non devono avere come principale obiettivo quello di fornire informazioni circa la loro condizione.
In questo senso quindi, analizzando il comportamento di alcuni software ci si è resi conto di come non siano presente alcun tipo di ottimizzazione.
Normalmente le richieste vengono eseguite senza curarsi del carico che queste generano sulla componente. Tale approccio può alterare il comportamento della rete ed è quindi da evitare.

In ogni sistema di monitoraggio lo scopo primario è quello di rilevare un guasto trascurando la prevenzione di un fault. Evitare un fault risulta spesso fondamentale soprattutto quando la componente monitorata è parte fondamentale della rete. Riuscire a rilevare situazioni pericolose prima di un guasto permette di:

- intervenire sul problema quando siamo in condizioni più stabili.
- evitare guasti irrimediabili per esempio, sotto il profilo hardware.

Intervenire in maniera proattiva su di un problema permette quindi di migliorare stabilità e longevità di una componente.

Altra osservazione interessante riguarda la natura dei controlli applicati alle metriche.

Nella maggior parte dei casi (come vedremo in dettaglio nello stato dell'arte) i controlli che si vanno ad eseguire su di una metrica sono molto semplici. Si fanno valutazioni sull'appartenenza ad un certo insieme o su funzioni matematiche tenute da un certo valore.
Non viene permesso di effettuare valutazioni incrociate prendendo in esame più di una metrica contemporaneamente al fine di diagnosticare fault. Si hanno quindi molte informazioni ma non le funzionalità necessarie a monitorare le correlazioni che queste hanno tra loro.
Relazionare tra loro più informazioni permette, per esempio, di capire quando una certa informazione ottenuta è affidabile. Possiamo infatti utilizzare il risultato di altre componenti per dedurre se il comportamento di un'altra unità monitorata è adeguato.

Osservando la tipologia di dispositivi connessi alla rete si sente il bisogno di non limitare al contesto di rete l'attività di monitoring.
Le reti attuali hanno al loro interno una varietà di dispositivi che spaziano da sensori RFID,rilevatori di temperatura,telecamere ecc. Stiamo parlando dell' Internet of Things (IoT).
Come analizzato nello studio di Gubbi,Buyya,Marusic e Palaniswami [13] l'interesse intorno a questo mondo sta crescendo costantemente e continuerà a farlo negli anni a venire. La difficoltà nell' includere questo tipo di componenti risiede soprattuto nel paradigma di interrogazione necessario.
Assume quindi fondamentale importanza avere la capacità di definire un modello che ci permetta di definire in modo semplice paradigmi di interrogazioni alternativi.

In tutta l'esperienza con i software di monitoring risulta fondamentale ottenere metriche in condizioni critiche.
Con condizioni critiche si intendono contesti in cui la componente monitorata non è in grado di garantire una risposta alle interrogazioni oppure in cui la rete non garantisce prestazioni scadenti. In questi casi, avere un sistema robusto, che non sia influenzato dalla situazione della rete è un requisito fondamentale. Si deve quindi essere in grado di registrare informazioni soprattuto in queste condizioni. Risulta oltremodo importante non andare in una condizione di stallo, evitando cioè di attendere per troppo tempo una informazione.
Sarà oggetto di successiva analisi capire quali siano le soluzioni implementate per raggiungere tale requisito.

Analizzando le informazioni raccolte da una interrogazione si nota come tutti gli stati siano composti da una singola informazione.
Solitamente quindi i software in commercio non acquisiscono e memorizzano informazioni se non quelle relative ad una specifica metrica. Sebbene questa scelta semplifichi in modo notevole la definizione dei plugin pone delle limita-

zioni. Spesso infatti la necessità di poter vedere come una metrica correlata a quella da noi monitorata fosse ad un certo istante.
Altro svantaggio è riscontrabile quando è necessario monitorare metriche che non sono tabelle. Un caso ad esempio è quello delle tabelle di routing. In tale condizione risulterebbe interessante poter analizzare la tabella nella sua completezza.
In sintesi quindi monitorare utilizzando una singola metrica risulta limitante sotto vari aspetti: quello della correlazione dei risultati e della loro rappresentazione.

Nel prossimo capitolo si andrà a stilare in modo dettagliato quali saranno le risposte da noi richieste nell' ambito del monitoring.
I requisiti saranno divisi in due sezioni:funzionali e non funzionali.
Tali caratteristiche guideranno tutto il resto delle valutazioni che saranno fatte fino alle conclusioni.

# 3 Requisiti

Dopo questa breve panoramica sul mondo della fault detection, si presentano di seguito, requisiti che si vuole raggiungere nello studio della fault detection.
Le specifiche definite di seguito spaziano dalla rappresentazione dei dati alla definizione di funzionalità.
L'obiettivo di questa sezione è quindi circoscrivere in modo preciso gli obiettivi da raggiungere.
Con le informazioni qui raccolte si andrà poi ad analizzare lo stato attuale della fault detection.

## 3.1 Requisiti Funzionali

I requisiti funzionali sono elenchi di servizi che il sistema deve fornire[21].
Si partirà analizzando i requisiti funzionali che ci prefissiamo di raggiungere.

### 3.1.1 Creare uno Storico delle Interrogazioni

In primo luogo si vuole creare uno storico.
Lo storico da mantenere deve essere in grado di contenere tutte le informazioni necessarie a tracciare il comportamento di un controllo.
Oltre a tracciare i risultati provenienti dalle interrogazioni,si andrà anche a memorizzare, in modo dettagliato, tutti gli errori risultanti dalla loro esecuzione.
Al fine di non immagazzinare dati ridondanti, si tracceranno le sole ¨differenze¨che intercorrono tra le varie esecuzioni di un controllo.
Infine, i dati saranno memorizzati in modo che siano facilmente analizzabili.
Sarà quindi fondamentale creare una struttura che risulti semplice per rappresentare tutte e sole le informazioni di cui si necessità.

### 3.1.2 Ottimizzare il Numero delle Richieste

Definendo un sistema di active detection assume particolare rilevanza ottimizzare il numero delle richieste verso una componente monitorata.
Questo è indispensabile per fare in modo che il sistema non sovraccarichi le componenti monitorate.
Per perseguire questo obiettivo si farà in modo che da una singola richiesta sia possibile ottenere più di una singola metrica.
Il fine è quindi quello di ottenere da una interrogazione l'intersezione dei dati necessari ai vari controlli.
In questo modo, inoltre, sarà possibile abbassare i tempi di esecuzione; un dato infatti non dovrà necessariamente essere reperito da ogni controllo.

Per quanto riguarda le applicazioni di rete, inoltre, raggruppare le richieste permette di diminuire il traffico generato.
Ottenendo molte informazioni da una interrogazione, non si vuole essere costretti a lavorare su dati di grandi dimensioni.
Avere informazioni inutili da analizzare complica il compito di un controllo, nonché le sue prestazioni.
L' obiettivo è quindi quello di definire un sistema che permetta di ottenere da una interrogazione i soli dati utili ad effettuare un certo controllo.
Questo oltre ad alleggerire la parte computazionale dei nostri controlli ci permetterà di renderne la scrittura più semplice.

### 3.1.3 Variare i Tempi di Interrogazione

Sempre al fine di ottimizzare le richieste un altro requisito importante è quello di variare i tempi di interrogazione.
Essendo partiti dal definire un sistema di interrogazione attivo, le richieste devono essere eseguite in modo intelligente.
L'idea di base segue le classiche dinamiche sociali di tutti i giorni: dobbiamo controllare meno spesso chi si comporta bene e tenere d'occhio più spesso chi è meno affidabile.
L' obiettivo a cui si vuole arrivare è far si che componenti con un comportamento adeguato non siano interrogate in modo intensivo, mentre, al contrario, siano richieste più informazioni possibili dalle componenti che hanno comportamento anomalo.
La variazione dell' intervallo di interrogazione dovrà garantire una risposta rapida ad i cambiamenti di stato.
Si vuole quindi, che un controllo adatti il suo tempo di interrogazione al comportamento di una componente, acquisendo la maggior parte delle informazioni in situazioni critiche.

### 3.1.4 Ottenere un Sistema Aperto

Sarà poi di fondamentale importanza definire un sistema aperto.
Un software di monitoraggio infatti deve essere in grado di adattarsi ad un numero elevatissimo di tecnologie disponibili.
Un sistema chiuso su se stesso tale da non permettere la sua estensione limita in modo consistente i campi di applicazione.
Si vuole quindi creare un' architettura che permetta di integrare nella nostra struttura plugin per l'interrogazione di ogni genere.
La struttura dei plugin dovrà essere tale da garantirne il riuso attraverso semplicemente attraverso l'uso di configurazioni diverse.
Al fine di rendere i plugin facilmente comprensibili sarà necessario definire una sintassi per:

- Capire da quali elementi deve essere composta la configurazione di un plugin.
- Riuscire a dedurre quali risultati produce.

La definizione di una sintassi comune sarà indispensabile al fine di rendere facilmente riusabili e condivisibili tali plugin.
Definito ciò sarà lasciata massima libertà su modo e tipo di dati da reperire.

### 3.1.5 Correlare Metriche delle Interrogazioni

La definizione di plugin con una fisionomia molto differenziata ci incentiva a correlare metriche.
L'intento è quindi quello di creare controlli che siano in grado di ricevere più di un singolo dato in ingresso.
In questo modo sarà possibile definire controlli che non basano la propria valutazione sull' appartenenza di un singolo valore ad un certo intervallo.
Il risultato di un certo controllo quindi potrebbe derivare dalla valutazione di informazioni ottenute prendendo in esame più metriche contemporaneamente.
Ad esempio ricevendo in ingresso lo stato delle porte di ciascun host nella nostra rete, si potrà dedurre se un certo ramo della nostra rete non funziona o se il problema è legato ad un singolo host.
Oltre a ciò si vuole poter confrontare valori presenti con rilevazioni passate.
Per rendere possibile ciò, sarà necessario avere uno storico capace di fornire tali informazioni in modo semplice.
In definitiva lo scopo ultimo sarà correlare un insieme di dati presenti e passati in modo non complesso sotto il profilo implementativo.
Anche in questo caso sarà lasciata la massima libertà sul numero di dati da analizzare contemporaneamente oltre che sul modo in cui essi possono essere correlati.

### 3.1.6 Valutare l' Andamento

Avendo dato risalto alla possibilità di correlare metriche si deduce come sia di rilevante importanza definire un metodo per valutare l'andamento di una componente.
Se in alcuni casi infatti una metrica assume valore positivo o negativo a seconda del risultato che rappresenta, non sempre questo è sufficiente.
La possibilità di avere valori multipli assegnabili al risultato di un controllo rende più facile agire in modo proattivo su di un guasto.
Questo, infatti, permetterebbe di valutare l'andamento di una componente, in modo da tracciare anche stati intermedi ad un malfunzionamento.
Questa possibilità risulta oltremodo utile anche a seguito di un fault. Al momento di esaminare le cause di un guasto, infatti potremmo analizzare quale

è stata l'evoluzione degli stati.
In questo modo potremmo escludere, ad esempio, rilevazioni che hanno avuto esito positivo dalla nostra analisi e concentrarci solo su quelle con valutazione negativa da parte del controllo.

Sarà necessario quindi:

- Avere un criterio di applicazione per la nostra scala di valutazione.
- Avere un modo per rendere tale valutazione obbligatoria.

Con questo requisito avremo quindi un utile strumento per analizzare in modo rapido lo stato di salute di una certa componente.

## 3.2 Requisiti non Funzionali

Si vanno di seguito ad elencare i requisiti non funzionali, quelli cioè che rappresentano i vincoli sui servizi offerti dal sistema[21].

### 3.2.1 Ottenere la Massima Robustezza

Tra i requisiti non funzionali poniamo in primo piano quello di ottenere un sistema robusto.
Tale caratteristica assume un ruolo fondamentale in un software di monitoring.
Esso infatti si trova a operare in condizioni complesse in cui possono verificarsi una grande quantità di errori.
Considerato poi che vorremmo ottenere più informazioni possibili quando una componente ha un comportamento anomalo, diviene fondamentale fare in modo che l' operatività sia sempre garantita.
Sarà quindi necessario:

- Gestire le eccezioni:errori generati da una interrogazione o da un controllo devono essere tracciati impedendo che danneggino l'esecuzione.
- Definire dei timeout:tempi di attesa troppo lunghi devono essere rilevati e risolti in modo da impedire che il sistema cada in stallo.

La gestione di queste situazioni dovrà essere estesa a tutte le componenti che non garantiscono un trattamento degli errori.

### 3.2.2 Massimizzare la concorrenza

Considerato il numero delle componenti da monitorare, che è solitamente grande sarà necessario massimizzare la concorrenza.
Nel definire una serie di operazioni parallele dovremmo porre particolare attenzione a quanto l'introduzione di nuovi thread incida in modo positivo

sulle prestazioni.
Considerato infatti che, si potrebbero avere più controlli che necessitano delle stesse informazioni, non è detto che sia sempre possibile parallelizzare.
Nel ambiente descritto, infatti, aumentare il numero di thread si aumenti non necessariamente aumenta la concorrenza.
Al fine di rendere la concorrenza efficiente, sarà necessario identificare casi e strategie per fare in modo che vi sia un reale miglioramento di performance.

### 3.2.3 Rendere il Sistema Facilmente Configurabile

Un altro obiettivo da perseguire durante tutta la nostra analisi sarà quello di rendere facilmente configurabile il sistema.
Un software che necessita di un numero troppo elevato di configurazioni va incontro a vari problemi.
Innanzitutto non incentiva la creazione di plugin e il loro utilizzo per il troppo tempo necessario al loro funzionamento.
Inoltre,causa notevole svantaggio nel caso sia necessario aggiornare una serie di informazioni riguardanti una componente già installata.
Per evitare tali limiti sarà necessario identificare quali siano le configurazioni utili e indispensabili a rappresentare in modo preciso le componenti monitorate eliminando tutto ciò che è superfluo o ricavabile in modi alternativi.
L' idea di rendere le configurazioni minimali dovrà essere alla base di ogni componente del sistema.

### 3.2.4 Integrabile con Sistemi Esistenti

Un requisito da perseguire è quello di ottenere un sistema integrabile.
Con tale termine intendiamo fare in modo che esso sia innestabile in qualsiasi contesto senza si vadano ad alterare prestazioni e comportamento della rete presente.
Inoltre vogliamo fare in modo che la struttura dei dati salvati permetta ad essi di essere esportati verso qualsiasi software.
Si pensi ad esempio ad una applicazione web che mostri i risultati acquisiti durante il monitoring.
L'integrazione si estende anche alla condivisione di informazioni verso software necessari ad attivare allarmi o sistemi di controllo.

Nel prossimo capitolo si introdurrà lo stato dell' arte nel campo del monitoring.
Osserveremo studi teorici che ci permetteranno di capire quali sono le basi teoriche su cui si basa il monitoring.

Si andranno ad analizzare poi alcuni esempi di software presenti sul mercato comparando come questi coprono i requisiti da noi descritti.

# 4 Stato dell' Arte

Attualmente tutta la letteratura sul "Network fault detection" ha le sue fondamenta in due approcci: uno passivo ed uno attivo.

## 4.1 Rilevare Guasti in modo Passivo

Si introdurranno di seguito quali sono i principi per la rilevazione di fault in modo passivo. I test di tipo passivo non introducono alcun tipo di richiesta o pacchetto sulla rete.
L'analisi viene portata avanti osservando l'evoluzione di ingressi ed uscite di una componente. Il tutto viene portato avanti senza che l'unità monitorata debba preoccuparsi di rispondere a richieste circa il suo stato.
Il vantaggi di tale approccio, risiedono nell' overhead nullo generato sulla rete. La rete su cui opera un sistema di tipo passivo non risente minimamente della sua presenza.
Un primo approccio teorico al passive monitoring è descritto nel lavoro svolto ai Bell Laboratories [14] e in maniera simile da Miller [17] e Ural [23].
In tali studio si analizza il comportamento di una componente al fine di realizzare una macchina a stati finiti (FSM) che permetta di rilevare, osservando input ed output, situazioni di errore della componente.
I risultati ottenuti sono comunque di basso livello, l'implementazione di una FSM risulta complessa (soprattutto per macchine non deterministiche). La caratteristica interessante di questo studio è la trasparenza ottenuta, in tutta la rete una volta creato la FSM niente è stato cambiato.
In un approccio più pratico solitamente quello che viene fatto è analizzare il traffico di una serie di componenti al fine di trarre conclusioni sullo stato.
Un interessante esempio accademico, è quello ideato da Agarwal [1] il quale propone un architettura totalmente trasparente per l'analisi del traffico.
Nella soluzione proposta, host riceve il traffico passante per la rete, viene "istruito" su quale traffico analizzare mediante dei pacchetti speciali detti Activation Packet e da quel momento raccoglie autonomamente informazioni.
La ricerca svolta da Agarwal pone le sue basi su altri software di monitoring i quali hanno il compito di catturare ed analizzare il traffico senza però fare nessun tipo di detection.
Il grosso limite della passive detection risiede nell' approccio black box che implmentano.
Analizzare infatti il comportamento di una componente dall'esterno non permette di trarre avere informazioni complete su di essa.
Solitamente quindi la passive detection deve essere affiancata anche da una parte attiva al fine di dare una visione completa dello stato. Sono molti gli studi che portano avanti questo approccio ibrido come ad esempio quello di Lowekamp[15].

## 4.2 Le Tecniche di Rilevazione Attiva

Un secondo metodo per rilevare fault nella rete è quello di tipo attivo.
L' architettura di un sistema attivo, solitamente è composta da uno o più probe (posizionati in punti strategici della rete) i quali inviano verso i dispositivi connessi alla rete delle richieste circa il loro stato.
Dopo aver rilevato una condizione anomala, è possibile ricavarne le cause andando a contattare direttamente la componente interessata.
Le criticità di un approccio di tipo attivo sono essenzialmente due:

- Generano traffico sulla rete, non utile alla sua operatività ma solo al suo monitoraggio.
- Hanno un costo per la componente monitorata, essa infatti dovrà usare parte del suo tempo per rispondere a richieste sul suo stato.

La maggior parte delle ricerche condotte in questo campo hanno il compito di limitare il numero di richieste.
Un primo esempio è riscontrabile nella ricerca condotta da Lu Lu [16] il quale propone una divisione in "stages" per analizzare lo stato della rete. Ogni stage quindi controlla solo una parte dei nodi, proseguendo attraverso ogni fase si ottiene una copertura totale dei nodi, generando un traffico minore sulla rete. L'idea è quindi quella di controllare più spesso le componenti comprese nei primi stage raggiungendo comunque la copertura totale, solo dopo una serie di passi intermedi.
L' altra idea interessante su cui si basano molti studi sull' active probing è quella di avere delle dipendenze tra le componenti. Tale caratteristica, permette, in caso di fault, di analizzare rapidamente i nodi influenzati dal fallimento di una certa componente.
Una via comune per perseguire tale obiettivo, sfruttata sia da Lu Lu [16] che da Chu L.W. [6], consiste nel rappresentare le dipendenze tramite Reti di Bayes . Una rete di Bayes [25] è un modello grafico che rappresenta un insieme di variabili correlato delle dipendenze usando un grafo aciclico diretto, i nodi del grafo rappresentano variabili casuali.
Con questa struttura si rappresenta quindi:

- con un nodo una componente di rete
- con un arco il legame ( con relativa proabilità) tra due componenti.

Con questa rappresentazione risulta facile capire quali sono le componenti interessate da un certo guasto.
L' uso di metodi attivi per la rilevazione di fault è molto diffuso. La diffusione di questa tecnica è favorita anche dall' aumento delle potenzialità offerte dai disposivi di rete. Come analizzato da tennenhouse [22] molti dei dispositivi presenti sulla rete sono ormai programmabili.
Questo permette di delegare alle componenti monitorate parte dell'analisi

sul loro stato, riducendo quindi l'impatto dell' active monitoring sulla rete.

## 4.3 Soluzioni per il Monitoring Presenti sul Mercato

Si presenta di seguito una serie di software che rappresentano una parte delle soluzioni in commercio per la fault detection.
L' architettura di questo genere di software prevede la presenza di una macchina la quale funge da server nella rete.

### 4.3.1 Zabbix

Zabbix è un software open source per il monitoring con una struttura molto complessa ed una serie di caratteristiche che lo rendono molto versatile.
I file di configurazione sono elencabili in cinque categorie [26]:

- Host o Gruppi: permettono di definire un host o un gruppo di essi da monitorare.
- Trigger: definiscono quali sono le regole da applicare ad un determinato servizio in modo da classificarlo con uno dei due stati ammessi: OK o PROBLEM.
- Event: sono generati a seguito di un cambiamento di stato di un trigger.
- Notification: definiscono messaggio e chi deve essere contattato a seguito di un evento.
- Template: semplificano le configurazioni, possono essere applicati ad una serie di host, la loro modifica si ripercuote su tutti gli host (o gruppi) a cui sono applicati.

Risulta interessante la possibilità di salvare in un database(MySQL, PostgreSQL, SQLite) lo storico dei dati acquisiti. Viene data la possibilità di configurare come questi dati devono essere memorizzati.
Per analizzare in maniera più comoda alcuni tra i più comuni servizi ,viene fornito un agent che installato sulla macchina si occupa di raccogliere le informazioni definite ed inviarle al server Zabbix.

Osserviamo come non sia disponibile una piattaforma di debugging delle configurazioni.
La struttura di Zabbix lo rende complesso da configurare quando si ha la necessità di monitorare lo stesso servizio per istanze diverse.
Per tale situazione infatti i template non agevolano la configurazione.

Confrontando come questo si pone rispetto ad i nostri requisiti, notiamo come sia possibile correlare metriche.
Risulta possibile, scrivere espressioni per i trigger [27] incrociando dati provenienti anche da istanze passate del controllo.
Non è invece possibile variare i tempi di interrogazione a seconda del comportamento della componente.
Zabbix permette solo di impostare delle parti del giorno in cui una certa componente non viene monitorata.
Infine si ha la possibilità di creare uno storico e ottenere dati da esso.

### 4.3.2 Nagios

Nagios è un software sviluppato da Nagios Enterprises focalizzato sul monitoraggio di rete.
Esso è in grado di effettuare entrambi i tipi di detection:passiva ed attiva.
Si ha la possibilità di configurare trap che il programma riceve da dispositivi di rete e comandi per interrogare attivamente componenti della rete.
La grande forza di Nagios risiede nell' essere un sistema aperto; permette di monitorare ogni genere di parametro anche non strettamente legato ad una caratteristica di rete (si pensi ad esempio alle temperature dei dischi su di un server).

Si analizza di seguito la struttura di Nagios.

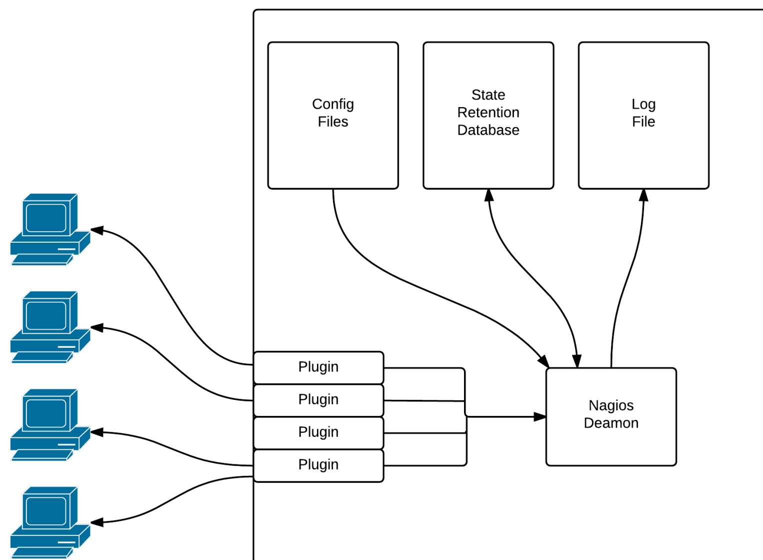



Figura 1: Architettura Nagios [3]

Essa è composta da:

- File di configurazione: attraverso una serie di file di configurazione vengono definiti

- host
  - gruppi
  - comandi
  - servizi
  - ecc

- State retention database: se attivato permette a Nagios di salvare l'ultimo stato di un host prima di essere terminato e ricaricare questo valore una volta avviato.

- Log File: contengono lo storico degli errori rilevati da Nagios.

- Plugin: sono definiti in vari linguaggi e permettono di aggiungere estensibilità a Nagios.

Sebbene Nagios rappresenti un software di grande successo presenta limitazioni dovute alle scelte fatte sulla sua architettura [18].
Come analizzato da Mongkolluksamee e Sophon[19], la forte flessibilità sulla quale si basa il software ha vantaggi e svantaggi. Questa infatti permette la definizione di infinite configurazioni, consentendo quindi di adattarsi ad una infinità di situazioni.
Lo svantaggio risiede nella complessità delle configurazioni,le quali non sempre risultano di semplice definizione. Il numero elevato di configurazioni, inoltre, rappresenta un ostacolo nei casi in cui si ha un numero basso di servizi/host da configurare. In tali situazioni si ha un gran numero di file di configurazione per monitorare poche componenti.
Altro aspetto limitante è dato dalla GUI esposta da Nagios.
L' interfaccia offerta è fin troppo minimale, non permette di avere una visione immediata dello stato della rete.
Infine Nagios manca di un sistema per l'archiviazione dello stato in modo nativo.
Sono presenti una serie di file di log ma questi risultano utili soprattutto per analizzare lo stato attuale della rete.
Per porre in parte rimedio a ciò è stata sviluppata un add-on per esportare dati da Nagios verso un database MySQL.

Le NDOUtils [11] sono utilizzate per esportare dati da Nagios verso sorgenti esterne.
La struttura prevede la presenza di un Event Broker il quale estrae i dati dal demone di Nagios di diverso genere (configurazioni o eventi a runtime). I dati possono vengono inviati tramite un file, un socket Unix o TCP al demone NDO2DB il quale si occupa di convertire i dati ricevuti dal NDOMOD ed inserirli in un database.
In questo modo è possibile effettuare operazioni di analisi più complesse sulle

informazioni catturate.
Al fine di avere informazioni anche antecedenti all' attivazione delle NDOUtils è possibile far convertire al modulo NDOMOD i file di Log presenti in Nagios per inviarli al NDO2DB.

Per chiudere l'analisi di Nagios, raffrontiamo le sue caratteristiche con i requisiti.
Non esiste un sistema efficiente per creare uno storico. Sebbene siano presenti le NDOUtils queste, oltre ad essere complesse nella loro struttura, non permettono di utilizzare tali dati direttamente in nagios.
Non è possibile correlare metriche. I controlli si limitano a controllare se un valore è all'interno di un certo intervallo.
Non si ha la possibilità di tempi di interrogazione, più precisamente si ha la possibilità di scegliere due possibili valori di polling: uno in caso di risultati positivi ed uno per quelli negativi.
Non si ha nessun sistema per ottimizzare le richieste verso una certa componente.
Infine si compre il requisito di avere un sistema aperto con la possibilità di aggiungere nuovi plugin.

## 4.4 Il Monitoring Come Servizio Cloud

Come analizzato da Reid [20], i servizi di tipo cloud stanno interessando un numero sempre maggiore di ambiti.
Anche il mondo del monitoring ha visto nascere software di monitoring di tipo SaaS(Software as a service).
Come descritto da Vaquero [24] un SaaS permette di condividere un servizio necessario ad una grande quantità di utenti attraverso una struttura di tipo cloud.

La struttura di questi servizi è composta principalmente da:

- una serie di agent per diverse ricevere dati da diverse piattaforme
- una serie di plugin per controllare le applicazioni più diffuse
- un interfaccia web per controllare le rilevazioni anche nel passato
- una serie di API per definire proprie applicazioni da controllare

Non è necessario quindi, configurare una macchina server ma è sufficiente installare su diversi dispositivi un agent. Un agent è un piccolo software che ha il compito di inviare i dati acquisiti verso il cloud server.
La configurazione viene effettuata tramite un interfaccia web che permette di definire quali tipi di controlli devono essere effettuati tramite un certo

agent.
Le metriche disponibili sono solitamente predefinite dal fornitore del software e non permettono di dichiararne altre personalizzate.

### 4.4.1 Datadog

Datadog offre un servizio di cloud monitoring che consente di monitorare i gli elementi di rete su cui è installato l' agent [9]. L' agent è disponibile per diverse piattaforme, e permette di monitorare una grande quantità di servizi monitorabili.
Linterfaccia lato server permette di definire:

- Definire Host (questi vengono automaticamente aggiunti una volta che viene installato lagent sull host in questione).
- Definire trigger.
- Configurare la gestione delle notifiche.

Il servizio mantiene uno storico dei dati rilevati per un tempo definito.
Vengono salvate tutte le rilevazioni effettuate su di una certa componente.
I risultati delle interrogazioni possono essere comparati sfruttando grafici;è possibile confrontare più host contemporaneamente visualizzando un grafico temporale.
I trigger utilizzano delle metriche interessanti, permettono di impostare situazioni di allarme che utilizzano lo storico delle rilevazioni.
Ad esempio possono essere fatte valutazioni sulla media di un valore in un certo range di tempo.
Un altra features interessante è quella viene chiamata Outlier detection [10].
Essa, consente di confrontare l'andamento di un particolare valore monitorato con un gruppo di host simili.
Se tale valore si discosta troppo dal comportamento generale degli host viene rilevato.
Infine Datadog non permette di definire dipendenze tra host, o tra servizi.

Affiancando Datadog ai requisiti precedentemente stilati si nota come sia mantenuto uno storico.
Si ha la possibilità di correlare metriche confrontando anche dati passati.
Inoltre, l'outlier direction, rappresenta una primitiva analisi su insiemi di dati provenienti da host diversi.
Non si ha la possibilità di variare i tempi di interrogazione.
Infine l' integrazione con sistemi già presenti risulta limitata. Si hanno pochi sistemi di notifica che, per ragioni archietturali, non possono essere interni alla nostra rete.

### 4.4.2 TrueSight Pulse

TrueSight Pulse è un software sviluppato da BMC.
La sua architettura è simile a quella di tutti i software SaaS per il monitoring. Viene fornito un agent da installare sulle componenti da monitorare, che, una volta avviato invia ad i server dell' azienda una serie di metriche. Si ha la possibilità di definire metriche personalizzate sfruttando le API messe a disposizione da BMC [4]. Esiste poi una via alternativa (non ufficialmente supportata) che permette tramite il plugin Boundary Shell Plugin [5] di utilizzare il ritorno di uno script qualsiasi per effettuare delle metriche.
Anche in questo caso i dati salvati sono rappresentati dal solo valore risultante dall' interrogazione.
Non è permesso poi definire dipendenze tra host, o tra servizi.

Confrontando le caratteristiche di TrueSight con i requisiti, si nota come esso crei uno storico. Tale storico permette di monitorare l'andamento di una certa componente nel tempo attraverso grafici.
Il sistema non è facilmente estendibile. Viene descritta un procedura per definire nuovi plugin che aggiornino lo storico ma essa risulta poco intuitica.
Non è possibile in modo alcuno variare i tempi di interrogazione.
Infine, sebbene siano offerti molti servizi l'integrazione è legata a servizi esterni alla nostra infrastruttura.

## 4.5 Uno Sguardo Riassuntivo sulle Soluzioni Presenti

Al termine di questa analisi dello stato dell' arte risulta interessante fare alcune condiderazioni.

| | Datadog | T.S.Pulse | Zabbix | Nagios |
|---|---|---|---|---|
| Creare uno storico | ✓ | ✓ | ✓ | |
| Ottimizzare le richieste | ✓ | ✓ | | |
| Sistema Aperto | ✓ | | | ✓ |
| Correlare Metriche | ✓ | | ✓ | |
| Valutare Andamento | | | | ✓ |
| Massimizzare la concorrenza | ✓ | ✓ | ✓ | ✓ |
| Sistema Resistente | ✓ | ✓ | ✓ | ✓ |
| Configurazione semplice | | ✓ | | |
| Variare tempi interrogazione | | | | |
| Integrabile | | | ✓ | ✓ |

In primo luogo si osserva come il requisito ottimizzare le richieste è soddisfatto solo dai software di tipo cloud.
I software di monitoring classici non hanno infatti nessuna politica per ottimizzare le richieste. L'uso degli agent invece, permette di delegare parte dell'analisi alla componente monitorata, diminuendo il carico sulla rete.
In secondo luogo è interessante notare come la definizione di un sistema resistente sia una condizione fondamentale e affrontata egregiamente da tutti i software presi in esame.
Infine osserviamo come li requisito massimizzare il parallelismo sia soddisfatto da tutti i software presi in esame. Questi infatti possono aumentare la loro capacità di interrogazione, aumentando il numero di poller attivi contemporaneamente.

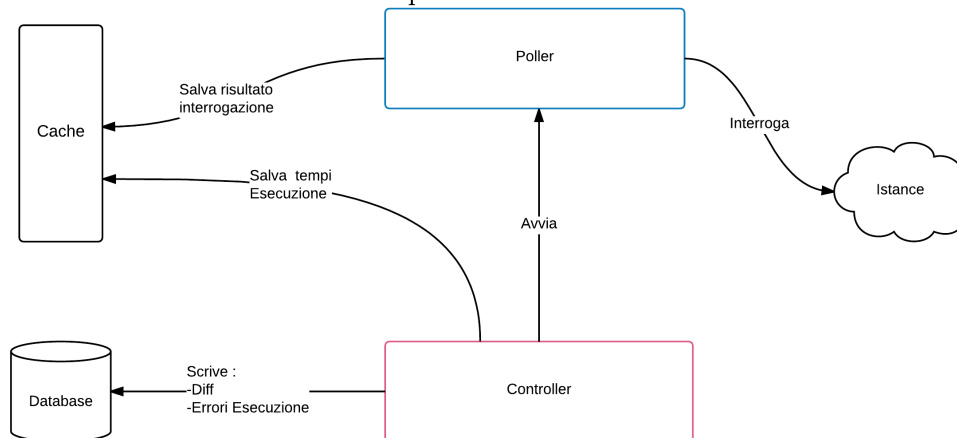
Nel prossimo capitolo si analizzerà l'architettura della soluzione utilizzata per raggiungere la copertura dei nostri requisiti.
Si metterà in luce per ogni scelta fatta quali requisiti si vanno a soddisfare.

# 5 Architettura della Soluzione

In questa sezione si analizzano in dettaglio le idee che hanno portato alla definizione della nostra architettura, per ognuna di esse a si descriverà l' impatto abbiano sui requisiti precedentemente definiti.
L'obiettivo del presente lavoro, non sarà, quindi descrivere la particolare implementazione che si è deciso di seguire, quanto piuttosto fornire tutti gli strumenti per capire quali siano i concetti che hanno guidato l'architettura.

## 5.1 Struttura Architetturale

Di seguito si descriveranno sia le entità che costituiscono il sistema il sistema sia le relazioni tra esse presenti.




Le strutture atte alla memorizzazione di dati come è ben visibile in figura sono due: una cache chiave-valore ed un database relazionale.

Si inizierà approfondendo il ruolo della cache nella nostra architettura.
La cache memorizza un elevato numero di informazioni utili al funzionamento del sistema: Tiene traccia del risultato delle interrogazioni (e del loro tempo di esecuzione), contiene le configurazioni per ogni componente, memorizza il tempo necessario al completamento di un controllo.
Riassumendo contiene due tipi di informazioni: quelle correlate all' acquisizione dei dati e quelle riguardanti le configurazioni.
Durante tutta la descrizione dell'architettura si osserverà come la cache risulti parte fondamentale in molti ambiti del sistema.
Tale importanza nasce da alcune esigenze necessarie al raggiungimento degli scopi prefissati, primo fra tutti la necessità di condividere dati tra più entità mantenendo un overhead basso per effettuare scrittura e lettura dei dati.
In casi simili a quello descritto,in cui si hanno pochi dati da scrivere ad ogni aggiornamento, l'uso di un approccio chiave-valore semplifica tutte le operazioni di scrittura.
La struttura necessaria a memorizzare un singolo dato mediante l'utilizzo

di una cache chiave-valore è minore, rispetto, ad esempio, ad un database relazionale.

Il primo componente che si analizzerà prende il nome di Poller.
Un Poller ha il compito di ottenere un certo insieme di informazioni, a questo viene quindi delegata tutta le gestione delle fasi di interrogazione verso la componente monitorata.
La caratteristica interessante dei poller risiede nella loro flessibilità.
Tale componente infatti può acquisire informazioni da più di una sorgente ed ottenere da esso un insieme di dati che non sia espresso dalla singola metrica ma da un insieme di esse.
Potendo quindi acquisire un insieme di informazioni, i poller tenderanno a massimizzare la quantità di dati risultati da una interrogazione; in questo modo, sarà possibile ridurre il numero delle richieste effettuate verso una certa unità per dati parte dello stesso gruppo.
La flessibilità dei poller si estende al tipo di dati che possono ottenere; essi infatti non sono vincolati ad elaborare informazioni relative ad apparati di rete ma, proprio grazie alla loro libertà di implementazione, permettono di catturare dati da una serie pressoché illimitata di sorgenti appartenenti agli ambiti più disparati.
Si potrebbero, quindi, incontrare nello stesso software, poller che acquisiscono informazioni sul regime di rotazione di un motore elettrico, affiancati da altri ottengono metriche riguardanti la tabella di routing di un server nella rete.
Il tutto senza al necessità di particolari caratterizzazioni nella loro definizione intervenendo solamente sulla definizione del poller.
La libertà dei poller si estende anche al modo in cui essi devono comunicare informazioni circa lo stato della loro esecuzione.
Si ha quindi la possibilità di tracciare in modo dettagliato la presenza di eventuali errori nell' acquisizione (andando a specificare in dettaglio la causa dell' anomalia) dei dati o specificare un valore di ritorno che indichi una situazione anomala al chiamante.
Descritta nel dettaglio la struttura dei poller si illustrano di seguito i requisiti soddisfatti da tale componente
Si osserva come il requisito introdotto nella sezione 3.1.2 sia strettamente connesso alla struttura dei poller.
Nel caso descritto nel presente lavoro è possibile affermare che, che questa caratteristica rientra nelle qualità da noi riscontrabili in essi, grazie alla possibilità di ottenere un insieme di dati.
Il secondo requisito soddisfatto è quello da noi introdotto nella sezione 3.1.4.
Come detto in precedenza, è possibile incrementare il tipo di dispositivi monitorabili e delle loro metriche, semplicemente definendo dei nuovi poller adatti allo scopo; questo ci permette di estendere le potenzialità del nostro sistema.

Il secondo componente che si approfondirà prende il nome di Controller.
Il controller ha un ruolo centrale nel nostro sistema, durante tutte le fasi del ciclo di acquisizione e controllo.
Ognuno di essi ha una serie di dipendenze che necessitano di essere soddisfatte per completare correttamente la sua esecuzione. Le dipendenze altro non sono che un elenco di Poller dai quali il controller necessita di ottenere informazioni aggiornate.
Per sapere se una informazione è recente il controller prima di procedere ad avviare di nuovo un poller accede alla cache nella quale per ogni poller è memorizzato l'ultimo tempo di esecuzione.
Se la differenza fra il tempo in cui viene eseguito il controllo e quello in cui è stato eseguito l'ultima volta il poller è minore o uguale a l'intervallo di esecuzione del controller, il risultato è da considerarsi attuale.
Questa scelta permette di soddisfare le richieste della sezione 3.1.2 attuando una strategia che non permetta di stressare una componente troppo spesso, per dati acquisiti di recente.
Per ogni poller nelle dipendenze, il controller può assumere tre diversi comportamenti:

- Se un altro controller sta eseguendo lo stesso poller, esso avvia una procedura di attesa, alla fine della quale, se il dato non è ancora disponibile, marca questo come non disponibile.
- Se invece il dato risulta già aggiornato, il controller accede alla cache e semplicemente lo preleva.
- Se il poller ha dati non aggiornati e nessun controller lo sta eseguendo, si procede all' esecuzione del poller al fine di aggiornare i dati.

Una volta ottenute tutte le dipendenze, se queste hanno avuto tutte esito positivo (cioè sono stati ottenuti dati aggiornati per ognuna di esse) il controller può procedere all' applicazione della funzione di controllo.
La funzione di controllo rappresenta una parte cruciale del Controller. Essa riceve in ingresso i dati delle dipendenze filtrate delle informazioni a lui superflue (si approfondirà in seguito come sia strutturato tale filtraggio) ed utilizza questi per determinare se ci siano,condizioni interessanti da tracciare.
Alla funzione di controllo viene lasciata molta libertà di implementazione, non ponendo limiti al numero di dati che essa può ricevere in ingresso.
Ad esempio, si potrebbero avere in ingresso dati riguardanti le schede di rete di un server e lo stato degli switch a cui esse sono connesse, in modo da capire da quale componente dipende il malfunzionamento di una porta.
Questo permette quindi di incrociare più dati al fine di prevedere situazioni che portano ad un malfunzionamento utilizzando un solo controllo.

Si immagini di avere un insieme di sensori di temperatura; tutti ritornano un valore inferiore di due gradi rispetto all' ultimo registrato. Sul singolo sensore questa notizia potrebbe non risultare di grande interesse ma il ripetersi sistematico su ognuno dei sensori da all'informazione tutt'altra rilevanza.
La funzione di controllo ha quindi lo scopo di trovare differenze significative da uno stato all' altro identificando quello che si definisce Diff.
Un diff è semplicemente una stringa che permette di capire cosa abbia portato un certo stato ad essere memorizzato in maniera permanente nel database.
Se ben implementato dalla funzione, il diff permette di avere un idea immediata di quale parte dello stato sia interessante in uno esso.
Da sottolineare come non sia possibile salvare nel database stati che non abbiano alcun diff specificato, questo rende possibile tracciare solo stati significativi per il controllo, evitando di andare ad affollare il database di dati simili e non significativi per identificare le cause di un malfunzionamento.
Inoltre, per ogni stato che ha un diff è necessario specificare una valutazione numerica per esso.
La valutazione di uno stato prevede quattro livelli:

- CRITICAL(1): lo stato viene ritenuto dal controllo un malfunzionamento grave per l'operatività degli elementi monitorati.
- SEVERE(2): lo stato è grave ma non tale da essere ritenuto un malfunzionamento, si applica a stati che potrebbero portare a situazioni di livello 1
- MODERATE(3): si applica nei casi in cui si hanno dei risultati che differiscono in maniera leggera dal comportamento canonico ma che al momento non rappresentano una situazione pericolosa.
- MINOR(4): lo stato rappresenta una condizione di buono stato in cui il controller ottiene uno stato che rispecchia quello atteso.

La definizione dei livelli si ispira alla scala AIS (Abbreviated Injury Scale) stilata da Civil, Ian e William Schwab [7] per dare una valutazione rapida di un trauma subito da un paziente.
La scala su 4 livelli, definita per il presente lavoro, oltre che a riconoscere quali stati sono gravi o no, permette di evidenziare quali di questi sono da tenere sotto controllo.
Se ben sfruttata quindi tale valutazione ci permette a seguito di un malfunzionamento di capire da quando il comportamento di una componente ha iniziato a divenire anomalo, oppure di intervenire in maniera proattiva su stati che stanno peggiorando pur non essendo ancora critici.
La funzione di controllo ha libertà nel decidere come contrassegnare uno stato, non si è obbligati a dare un casistica per ogni livello.
Ad esempio se il risultato è di tipo binario con solo uno stato buono ed uno cattivo questa sfrutterà solo il livello uno e quattro della scala.

Definiti quindi il diff e la valutazione ad esso correlata lo stato è pronto ad essere salvato in maniera permanente nel database. Dopo la terminazione della funzione di controllo il controller può considerarsi completato,viene quindi programmata la sua esecuzione dopo aver calcolato il suo nuovo intervallo di esecuzione.
L'intervallo di esecuzione con cui un controller viene eseguito viene influenzato da:

- Errori nell'acquisizione di dati da un poller
- Errori nell' esecuzione della funzione di controllo
- Valutazione di livello basso assegnata dalla funzione di controllo
- Corretta acquisizione dei dati delle dipendenze
- Valutazione positiva della funzione di controllo

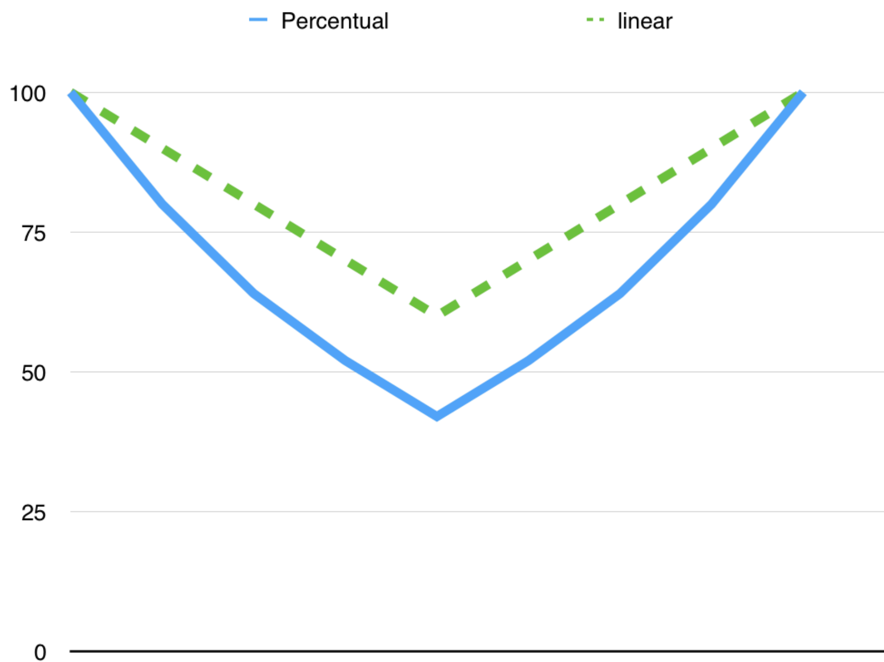
Si hanno due possibili alterazioni dell' intervallo di esecuzione.
Per i primi tre casi cioè casi di situazioni da considerare negative, si procede a diminuire l'intervallo.
In questo modo si ottengono più informazioni possibili sulla condizione anomala che il controller sta rilevando.
La formula applicata ha la seguente forma:

$$interval = interval - (10\% \quad of \quad interval)$$

Per i casi in cui invece si hanno responsi positivi dall' esecuzione del controller in tutte le sue parti, si procede ad aumentare il tempo di interrogazione.
Nel caso in cui il controllo si comporti in modo conforme a quello atteso, il numero di stati da acquisire potrà quindi diminuire.
L'intervallo di esecuzione viene quindi alterato nel modo seguente:

$$interval = interval + (10\% \quad of \quad interval)$$

Si osserva come entrambe le formule effettuano incrementi di tipo percentuale.
Un incremento di questo tipo permette di reagire rapidamente alle situazioni anomale ed essere più cauti nel ritenere di nuovo affidabile una certa componente.




Nell' esempio sopra presentato, sull' ascissa rappresentiamo i punti di cambiamento dell' intervallo, mentre sulla ordinata sono riportati i risultati ottenuti in secondi.
Il contesto descritto rappresenta l'evoluzione degli intervalli dopo che è stato rilevato uno stato di errore(l'intervallo scende da 100 a 42 secondi) e il comportamento dopo che il controllo ha ripreso a completare la sua esecuzione in modo positivo (l'intervallo risale da 42 a 100 secondi).
Comparando il comportamento percentuale con il caso lineare è possibile osservare come quest'ultimo non risulti essere il più adeguato per gli scopi del lavoro presentato.
D'altronde, La curva lineare risulta essere è troppo lenta a scendere nei momenti immediatamente successivi alla situazione anomala e troppo veloce nel risalire al valore iniziale, dopo che si sono ricevuti riscontri positivi (ottenendo quindi un numero minore di stati rispetto al caso percentuale).
In definitiva possiamo dire che il caso percentuale garantisce una diffidenza maggiore soprattutto nei casi in cui il controllo torna ad avere un comportamento adeguato.
Ovviamente la formula che va ad alterare l'intervallo di esecuzione ha un limite superiore di crescita.
In questo modo, l'intervallo non aumenta troppo anche se si hanno riscontri sono positivi per tempi.
Si ha poi un limite inferiore, definito nella configurazione del Controller, il quale rappresenta inoltre l'intervallo iniziale.
L' esecuzione di un controller, prevede di tracciare tutti i possibili errori ri-

scontrabili durante la sua esecuzione, definendo un diff per ogni dipendenza che non va a buon fine.
Da notare come il processo di aggiornamento delle dipendenze non venga interrotto anche a seguito di errore.
Tale strategia permette di avere informazioni su quali poller non sia possibile aggiornare; ovviamente, non sarà possibile eseguire la funzione di controllo poiché richiede di avere tutte le dipendenze soddisfatte.
Oltre a ciò il controller, per ogni dipendenza non soddisfatta, procede a porre in esecuzione il prima possibile tutti i controlli interessati a tale dipendenza, in modo che essi non perdano stati interessanti per la componente a causa di un intervallo di esecuzione troppo ampio.
Si approfondirà nei paragrafi successivi come, tale gestione degli errori, incida sulla robustezza del sistema, nella quale i controller hanno un ruolo chiave.
Avendo un quadro definito di come sono strutturati i controller è possibile affermare che il punto 3.1.5 è soddisfatto, permettendoci di collegare più dati in un unico controllo è pienamente soddisfatto dal controllo.
L' obbligatorietà di assegnare una valutazione, per salvare uno stato nel database permette di valutare l'andamento di un certo controllo.
La definizione dell' incremento percentuale,le cui caratteristiche sono state precedentemente analizzate, permette al requisito 3.1.3 di essere soddisfatto avendo inserito un metodo che assicura una reazione rapida ad i cambiamenti di stato.

Di seguito si deriverà il ruolo del Database inserito nella nostra architettura.
Come intuito dalla descrizione delle componenti fatta fino ad ora, esso immagazzina un insieme di informazioni risultanti dall' esecuzione di un controller.
Le informazioni contenute nel database sono quindi tutte quelle necessarie ad avere un quadro completo dello stato salvato:

- Id: un identificativo numerico per lo stato.
- Controller:nome controllo da cui deriva lo stato.
- Istance: nome dell' istanza del controllo, potremmo vederla come una particolare configurazione del controllo si analizzerà meglio in seguito di cosa si tratta.
- Timestamp: data e ora a cui uno stato fa riferimento.
- Diff: diff risultante dal controllo.
- Full status:Nel caso la funzione di controllo sia stata eseguita raccoglie tutti i dati che questa ha ricevuto in ingresso.

- Level: rappresenta la valutazione assegnata allo stato secondo la scala da noi precedentemente stilata.

Questo insieme di informazioni ci garantisce per ogni stato un insieme corposo di dati, tale da delineare una fotografia dettagliata di cosa è risultato dall' esecuzione di un particolare controllo.
Sotto il profilo prestazione le scritture su di un database sono ovviamente più onerose rispetto a quelle nella cache.
Tale costo viene comunque ammortizzato dal numero complessivo di scritture nel database; esse sono in numero molto minore rispetto alla cache.
Nel database vengono salvati stati che hanno un diff (mentre la cache viene continuamente aggiornata).
Al fine di non aggiungere informazioni superflue e ridurre i dati salvati vengono registrate le sole informazioni utili all'esame del controller.
Il database non si limita a memorizzare dati in maniera permanente, esso permette alla funzione di controllo di accedere ad esso per effettuare controlli che prendano in considerazione anche dati storici.
Si ha la possibilità di richiedere un numero arbitrario di stati antecedenti a quello attuale non necessariamente relativi allo stesso controllo che effettua la richiesta.
Il database va quindi a coprire il requisito per noi fondamentale descritto nella sezione 3.1.1, ponendo attenzione a non memorizzare dati superflui e minimizzando l'occupazione dei dati necessaria.

In ultimo si esaminerà il concetto di instance.
Tale concetto si applica in modo simile a Poller e Controller.
Una instance rappresenta una astrazione di una componente monitorabile, può rappresentare una singola istanza di una componente o un gruppo di esse.
Per i poller una instance altro non è che una particolare configurazione da applicare alla funzione di interrogazione.
Una instance può definire al suo interno più di una configurazione per lo stesso poller.
Questa strategia risulta interessante per creare gruppi di componenti dei quali si monitorano le stesse caratteristiche.
Nei controller ritroviamo il concetto di instance sotto forma di configurazione.
Si ha quindi la possibilità di creare più istanze dello stesso controllo.
Un controller dovrà poi specificare quale instance applicare ad un determinato poller.
In questo modo è possibile definire più istanze dello stesso controller che operano su dati diversi senza dover scrivere codice ulteriore.
Le instance e le configurazioni dei controller sono contenute nella cache.
L'uso delle instance, risulta utile per il punto 3.2.3, rendendo più semplice

la definizione di nuove istanze simili e favorendo il riuso del codice.

## 5.2 Il Filtraggio dei Dati Attraverso i Selettori

Durante tutta la descrizione dei controlli abbiamo tralasciato il modo in cui si intende rappresentare e manipolare insiemi di dati in modo utile ai controlli.
Da una interrogazione si vuole estrarre un insieme di oggetti che sono uguali per le informazioni necessarie a rappresentarli.
Con le considerazioni fin qui fatte si è deciso di utilizzare una astrazione che ben racchiude le esigenze emerse: una tabella.
Una tabella per sua definizione organizza dati in righe e colonne.
Nel caso specifico:

- Le colonne: rappresenteranno i vari campi che servono ad identificare un elemento dell' insieme.

- Le righe: rappresentano per ognuna di esse le varie istanze dell' insieme.

Un poller al termine della sua interrogazione ritorna quella che concettualmente è una tabella e che nella nostra architettura prende il nome di stato.
L' obbligo di definire una struttura tabellare per la rappresentazione dei dati fa si che, al fine di ammortizzare il costo dovuto alla creazione di una tabella, sia vantaggioso ottenere un insieme di dati che sia maggiore della singola riga.
In questo modo si tende a scoraggiare la definizione di plugin che restituiscono un solo valore (per il quale sarebbe necessaria una tabella di una riga ed una colonna).
Al fine di rendere più circoscritto il tipo dei dati elaborabili dai controller, una tabella potrà contenere solo: stringhe e quantità numeriche.
Se avere un tipo che permetta di rappresentare quantità è indispensabile per garantire ai controlli di effettuare confronti tra valori in modo diretto, con l'introduzione delle stringhe tra i tipi di dato si lascia la libertà di rappresentare anche dati non strutturabili sotto forma numerica, permettendo quindi di interpretare questi nel modo che i controlli ritengono più adeguato.
Il contesto in cui si va ad elaborare i dati è quindi così definito: si hanno poller che raccolgono stati che preferibilmente sono di grande dimensione e vengono consumati da più di un controller.
In questo scenario si è reso necessario definire uno strumento per fare in modo che ogni controllo non si trovi ad elaborare una quantità di dati a lui superflua a portare a termine la funzione di controllo, introduciamo i selettori.
Al fine di comprendere quale sia l'idea che sta dietro alla definizione dei selettori, si utilizzano due semplici nozioni dell' algebra relazionale.

Il primo operatore è quello di proiezione.
Riprendendo la definizione data da Ghelli,Albano e Orsini [2] la proiezione si definisce come:

$$\pi_{A_1,A_2...A_m}(R) = \{t[A_1, A_2...A_m]|t \in R\}$$

In cui R è una relazione, $A_1...A_m$ attributi di R.
Applicando questo operatore ad una relazione, questo restituisce quindi un sottoinsieme $\{t[A_1, A_2...A_m]|t \in R\}$ degli attributi di R.
Si hanno quindi tutte le ennuple originarie ma con un numero di colonne minore uguale all' insieme originale.
Il secondo operatore che si va a introdurre è quello di Restrizione (anche detto di selezione); la sua definizione può essere racchiusa in questa breve scrittura:

$$\sigma_{\Phi}(R) = \{t|t \in R \wedge \Phi(t)\}$$

Anche in questo R rappresenta una relazione, mentre $\Phi$ rappresenta una condizione sugli attributi di R.
Applicato ad un relazione quindi, l'operatore di restrizione restituisce tutte le ennuple che fanno parte di R e soddisfano la condizione $\Phi$.

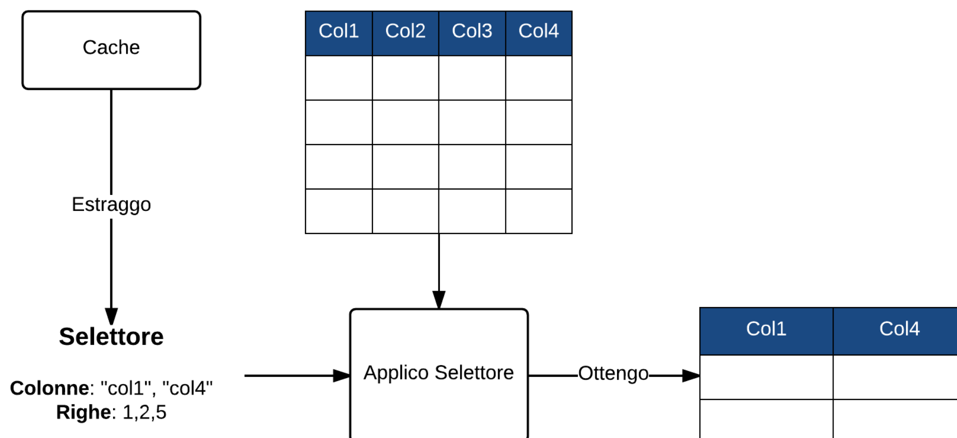
In primo luogo notiamo come il concetto di relazione sia comparabile a quello di stato nel presente sistema.
Con queste due definizioni si procede a definire il concetto di selettore in modo formale sfruttando i due operatori sopracitati:

"Un selettore definisce la sequenza di due operazioni la prima di proiezione, la seconda di restrizione applicabili ad uno stato S".

Sebbene questa definizione risulti di immediata comprensione, necessità di qualche dettaglio in più.
La restrizione definita dall'algebra relazionale, infatti, non è la stessa che si va ad applicare con i selettori.
La restrizione che applica un selettore permette di specificare solo quali righe si è interessati ad ottenere.
Non viene quindi permesso di selezionare le righe attraverso una funzione delle ennuple.
La scelta di non avere una restrizione "pura" nasce dall' esigenza di semplificare la configurazione; definire una funzione per ogni selettore, infatti, avrebbe complicato la loro definizione nelle fasi di configurazione.
Esiste, infine, la possibilità di definire selettori che applichino solo la proiezione e non la selezione.
Questa possibilità utile in due casi: se si è interessati ad acquisire tutte le righe indipendentemente da quante esse siano, oppure quando si hanno stati

con un numero variabile di righe (si pensi ad esempio ad una tabella di routing in cui monitorare l'aumentare o il diminuire delle route è fondamentale).

Una volta definiti i selettori ben definiti a livello teorico andiamo a vedere come questi si integrano nell' architettura.
Un selettore è una configurazione memorizzata nella cache.
Ogni selettore al suo interno contiene due informazioni: il nome di ogni colonna che seleziona dallo stato, e le righe di esso a cui è interessato.
Un controller è obbligato a definire per ogni dipendenza un selettore da applicare allo stato ritornato.
I controlli sono quindi incentivati a prelevare dallo stato le sole informazioni interessanti all' esecuzione della funzione di controllo al fine di rendere questa più semplice e meno esosa sotto il profilo delle risorse.
La definizione dei selettori nella cache, inoltre, favorisce il riuso delle configurazioni da parte dei controller.




Nell' esempio sopra mostrato si evidenzia quello che il controller attua una volta che il processo di acquisizione è stato terminato con successo per ogni dipendenza.
Per ogni stato acquisito il controller procede a prelevare il selettore ad esso associato dalla cache e una volta estratto con successo, procede alla applicazione di questo allo stato completo estratto dalla cache.
Il risultato della applicazione è uno stato ridotto dall' applicazione delle regole scritte nel selettore.
Una volta terminata per ogni dipendenza la fase di applicazione dei selettori, viene passato l'insieme degli stati ridotti alla funzione di controllo.

## 5.3 Robustezza della soluzione

Come definito da Ghezzi [12] un programma si dice robusto se si comporta in modo accettabile anche in situazioni non previste dalla specifica dei requisiti.
La robustezza nell'architettura descritta ha un ruolo fondamentale. I casi più interessanti in cui il monitoring da noi implementato è determinante per prevedere o correggere una anomalia sono spesso instabili e di difficile previsione.
Solitamente infatti si incontrano latenze, impossibilità a reperire informazioni complete, dati danneggiati. Garantire l'operatività del sistema in questi scenari è quindi cruciale.

Il concetto di base evoluto poi nei in vari contesti dell' architettura per raggiungere tale obiettivo è quello di eseguire in un ambiente protetto tutte le operazioni che potenzialmente possono generare situazioni di errore.
Al fine di evitare comportamenti inaspettati,sia i controller che i poller non accedono direttamente alla cache ed al database ma sono obbligati a passare attraverso un livello intermedio. Questa libreria implementa al suo interno una serie di controlli il cui principale obiettivo è quello di catturare eventuali malfunzionamenti o errori nelle operazioni su di essi e fare in modo che essi non diventino critici per l' intero sistema. Oltre a questo porre un livello intermedio tra le basi di dati e chi le utilizza permette di aumentare la sicurezza sulle operazioni che controller e poller vanno ad effettuare su di esse.

Altro punto critico per la robustezza è senza dubbio l'esecuzione dei poller.
Essi infatti per loro struttura non hanno vincolo alcuno sulla gestione degli errori derivanti dall' acquisizione dei dati. Per questo motivo la funzione di acquisizione viene eseguita in una modo controllato in modo che eventuali eccezioni generate dal codice del plugin vengano intercettate.

Il secondo vincolo che devono rispettare i poller riguarda il tempo di esecuzione. Un poller ha un tempo massimo di dieci secondi per terminare la sua esecuzione, se non riesce a rientrare in tali tempistica la sua esecuzione viene interrotta.
In questo modo si garantisce che il sistema non vada in stallo per attendere il risultato di un poller.
Il Controller oltre a ciò esegue un controllo sul risultato salvato in cache al fine di verificarne la validità della struttura. In questo modo vengono bloccati dati ritenuti incompleti o mal formattati. Altro punto cruciale del processo di controllo è al momento dell'esecuzione della funzione di controllo; anche in questo caso essa viene eseguita in modo che non le sia possibile rilanciare eccezioni.

Un ruolo più marginale ma comunque utile per capire come il software si sta comportando è quello relativo a tutti i timestamp che vengono salvati durante l'esecuzione.
La presenza di una coda circolare che contenga le ultime 100 esecuzioni di un controllo con relativo tempo di esecuzione permette di sapere se il sistema stia ancora eseguendo controlli.

Tutte le funzionalità sopra descritte vanno quindi a coprire l'obiettivo di coprire il requisito 3.2.1.

Nel prossimo capitolo si introdurranno implementazione e validazione per l' architettura sopra descritta. Si discuteranno quindi strutture dati scelte, implementazione e istanze funzionanti del sistema.

# 6 Implementazione e Validazione

In questo capitolo si descrivono i problemi e soluzioni incontrati nella parte implementativa del architettura.Si mostreranno le scelte riguardanti le strutture dati e le tecnologie utilizzate per attuare le idee introdotte nell' architettura.

L'intero sistema è stato sviluppato sfruttando il linguaggio Python.
La prima sfida incontrata nella parte implementativa riguarda la rappresentazione dello stato. Il risultato concettuale che si vuole ottenere, come descritto nelle sezione 5.2, è quello di una tabella. Tale concetto viene poi affiancato dalla necessità di avere un formato compatto per la sua rappresentazione.Il formato scelto per la rappresentazione è quello JSON.
Lo scelta di tale tecnologia si basa su due motivi:

- Garantisce un formato abbastanza compatto per rappresentare i dati.
- Si presta bene a rappresentare una lista di dati dello stesso tipo.
- Risulta semplice effettuare diff tra due elementi.

La rappresentazione dello stato viene quindi ottenuta creando una lista di oggetti JSON.
Ciascun oggetto contiene un insieme di campi rappresentanti le colonne della tabella. Questa struttura permette di rappresentare in modo semplice e compatto uno stato.

| ipRouteDest | ipRouteDest | ipRouteType |
|---|---|---|
| "0.0.0.0" | "0.0.0.0" | 4 |
| "46.101.191.0" | "255.255.192.0" | 3 |

[ {"ipRouteDest":"0.0.0.0","ipRouteMask":"0.0.0.0","ipRouteType":4},
{"ipRouteDest":"46.101.191.0","ipRouteMask":"255.255.192.0","ipRouteType":3} ]

Nell' esempio sopra mostrato viene mostrato come una tabella di routing venga trasformata in formato JSON, come possiamo notare il risultato è compatto e leggibile allo stesso tempo.

Un secondo aspetto di cui si va a descrivere l'implementazione è quello riguardante la struttura scelta per gestire i Controller.
I controller sono gestiti tramite una lista ordinata per tempi crescenti, la loro

esecuzione è assegnata ad un pool di thread.
Tale pool viene creato all'avvio del software e rimane sempre attivo; si evita così di creare e terminare thread(essendo tale operazione costosa) incentivandone il loro riuso.
Al fine di controllare se un controller deve essere eseguito, ogni thread ha accede al prossimo tempo di esecuzione del primo controllo.
A questo punto:

- Se il tempo è maggiore del tempo attuale il thread andrà ad attendere per (tempo prossimo controllo - tempo attuale) secondi.
- Se il tempo è minore o uguale al tempo attuale il thread procede ad estrarre dalla lista il controllo e eseguirlo.

Il thread inserisce poi, in modo ordinato secondo il tempo del prossimo controllo, il controller solo al suo completamento.
In questo modo si evita che altri thread siano in grado di eseguire lo stesso controllo nel caso questo impieghi troppo tempo per essere completato.

Un altro punto fondamentale per fare in modo che tutto il sistema si mantenga operativo è quello della gestione degli errori. L' implementazione ottenuta permette di tracciare e gestire errori in modo da evitare che questi si propaghino fino a richiedere la terminazione del thread.
Al momento di richiamare una funzione di polling, si procede catturando l'eccezione di livello più alto sollevabile.
In questo modo, anche se la funzione di polling è implementata per non catturare nessuna eccezione da essa generata, siamo in grado di tracciare e contenere eventuali errori.
Tale strategia viene poi applicata in modo del tutto analogo nel caso della funzione di controllo.

Altro requisito fondamentale della nostra architettura è quello di evitare situazioni di stallo. Tali contesti sono facilmente riscontrabili nei contesti critici in cui andiamo a monitorare.
Per evitare ciò al momento dell' esecuzione di una funzione di poll si avvia un timer con timeout di dieci secondi. Se la funzione non ritorna alcun valore prima di tale tempo si solleva una eccezione e si termina l'esecuzione di questa.
Lo stesso criterio si applica alla funzione di controllo, in questo modo si cerca di scoraggiare la definizione di procedure di controllo troppo esose che potrebbero rallentare l'esecuzione del software.
Ricapitolando quello che abbiamo sviluppato è una sorta di "sandbox" che permette di catturare e tracciare: eccezioni e timeout.

Dopo aver definito la natura dei timeout si descrive come è articolato il

ciclo di attesa che un controller esegue nel caso un dato sia in esecuzione su di un altro controller.
Una volta rilevato che un altro thread sta eseguendo il poller, il controllo preleva e salva l'ultimo tempo di esecuzione per tale poller. Per cinque volte con un intervallo di due secondi, controlla se tale tempo è stato aggiornato. Se prima del termine del ciclo riscontra che il tempo è più grande di quello da lui acquisito procede a prelevare il dato, altrimenti traccia il poller come non disponibile.
Si noti come il tempo massimo del ciclo sia uguale alla soglia di timeout per le funzioni di polling.

Si evidenziano di seguito, alcuni interessanti dettagli riguardo l'implementazione dei poller.
Essi sono contenuti in una hashtable la quale garantisce un accesso pressoché immediato. L' accesso a tale struttura è regolato in modo mutuamente esclusivo. Un controller può conoscere lo stato di un controller accedendo ad un campo in esso contenuto chiamato `state` che può assumere due valori:

- EXEC: se il poller è in esecuzione da parte di un controller.
- NOEXEC quando il poller non è in esecuzione su nessun controller.

Grazie a questi due stati viene deciso il comportamento che il controller dovrà tenere per una certa dipendenza.
I poller contengono poi al loro interno una lista di controlli che sono interessanti alla sua esecuzione.
Tale lista viene sfruttata per mandare in esecuzione il prima possibile i controller interessati da un fallimento del poller.

Nella sezione 5.2 abbiamo descritto dettagliatamente il ruolo dei selettori. Analizzeremo adesso come essi siano effettivamente implementati ed applicati.
I selettori sono contenuti in una hashtable all' interno della cache. in questo modo sono estraibili in modo molto rapido. Ogni selettore è rappresentato da un oggetto json contenente due attributi: cols e rows.
Entrambi i campi sono delle liste, in cui la prima contiene i nomi delle colonne e il secondo l'indice delle colonne da selezionare.

```
"cols":["ipRouteDest","ipRouteType"],"rows":[0,1]
```

Il selettore sopra mostrato permette di selezionare le prime due righe e le colonne ipRouteDest e ipRouteType da uno stato compatibile.
Una volta ottenuti l'insieme degli stati completi per tutte le dipendenze, estrae per ognuno di questi il relativo selettore. Fatto ciò applica ad ogni stato il suo selettore tramite una apposita funzione che ritorna il nuovo stato. Sara l'insieme di questi risultati ad andare in ingresso alla funzione di controllo.

### 6.0.1 Riferimenti al Codice Sorgente

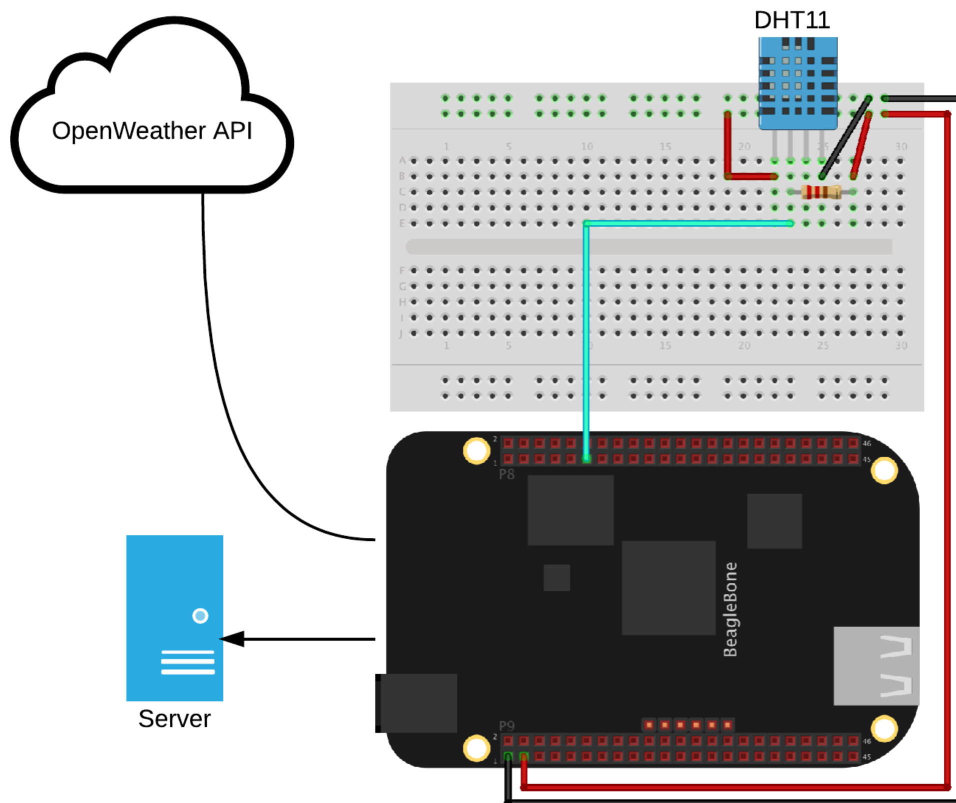
Si rimanda al codice disponibile online all' indirizzo https://github.com/lorenzobrunetti/Ntrouble, per ulteriori dettagli implementativi.

## 6.1 Validazione attraverso un Single-board computer

In questo paragrafo descriveremo un istanza del software sviluppato. L'intero sistema da noi ideato è stato messo in esecuzione su di un single-board computer.
Tra la miriade di soluzioni disponibili sul mercato, si è scelto quella proposta da Beagleboard tramite il modello BeagleBone Black[8].
Le motivazioni che hanno indirizzato su questo prodotto sono principalmente tre:

- Presenza di una porta ethernet dalle buone prestazioni
- Possibilità di espandere facilmente la capacità di memorizzazione tramite supporto microsd.
- Possibilità di utilizzare Python per accedere a tutte le componenti connesse ai pin.

Assume rilevanza significativa l'ultimo punto dell' elenco. La possibiltà di usare python anche per accedere a sensori e altre periferiche connesse alla beaglebone black, semplifica significativamente lo sviluppo dei plugin di polling.
In questo modo infatti è possibile scrivere direttamente nel codice del plugin tutto il necessario per l'interrogazione, senza richiamare eseguibili esterni.




Si noti come nell' installazione presentata siano presenti componenti di rete affiancate da altre che non hanno alcuna relazione con tale mondo.

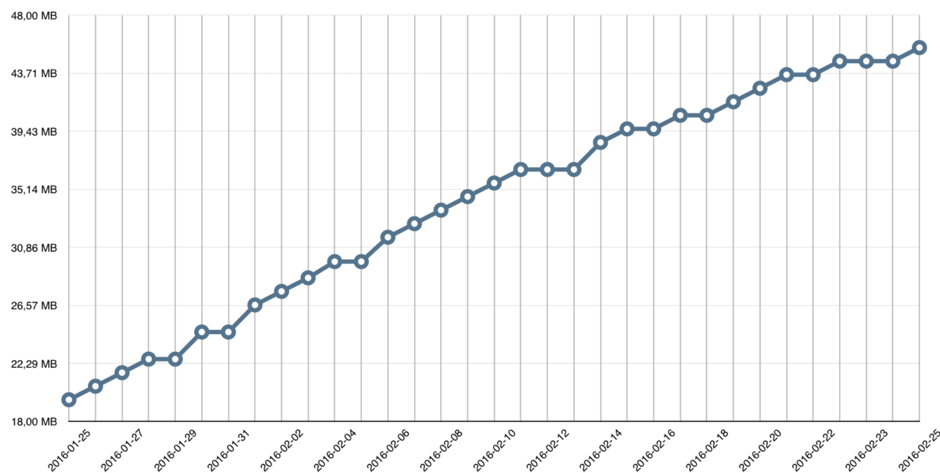
Si descrivono di seguito, i controlli che fanno parte della configurazione:

- file_modified_checker: confronta una serie di file di log al fine di tracciare quali di questi sono stati modificati.
- temp_checker: ottiene temperatura ed umidità tramite il sensore DHT11 e confronta i dati ottenuti con quelli nominali ottenuti tramite le API offerte da OpenWeatherMap.
- json_checker: effettua il diff di due json, nel nostro caso si tratta della tabella di routing di un server.
- webpage_checker: controlla se una certa pagina web si è aggiornata.

Di questi controller due risultano particolarmente interessanti.
Il controllo temp_checker infatti,integra dati provenienti da sorgenti molto diverse. è quindi possibile in grado di ottenere da sorgenti differenti informazioni riguardanti la temperatura al fine (per esempio), di sapere se la temperatura esterna è più alta di quella interna.

Il secondo controllo interessante è json_checker in quanto ci permette di apprezzare quanto il formato JSON sia ottimo per effettuarne diff. Tale controllo infatti traccia semplicemente i cambiamenti tra l'ultimo Stato acquisito e quello attuale. Questo controllo può essere quindi applicato a qualsiasi tipo di dato ritornato da un poller applicando una funzione di diff per json.

L'istanza sopra discussa è stata operativa dal 25/01/2016 al 25/02/2016.
Al fine verificare la bontà dell architettura, sul server monitorato è stato lanciato un script che disabilità per un tempo casuale la schede di rete. In tutta la sua esecuzione la board ha tracciato timeout ed eccezioni provenienti da tutti i plugin. Un impostazione dei tempi di controllo adeguati(come analizzeremo tra poco) al tipo di controlli, ha permesso inoltre di non perdere nessun controllo. In tale situazioni critiche, il sistema non ha mai terminato nessuno dei suoi thread.
Durante l'esecuzione la dimensione del database ottenendo i risultati sotto mostrati.




Si osserva come la dimensione del database in circa un mese di utilizzo sia stata in media di un megabyte al giorno. Tale risultato può essere considerato un successo, in quanto il database ha mantenuto in un intero mese di utilizzo, dimensioni ridotte.

## 6.2 Considerazioni sulle Prestazioni

In questa sezione si vanno ad elencare quali sono le prestazioni assicurabili dal sistema.
Tra i tempi costanti, entrano a far parte dell'esecuzione dei controlli sono:

- $t_{cg}$: tempo per ottenere un controllo dalla lista ordinata per tempo in cui essi sono contenuti.
  per come è strutturata la coda, è necessario accedere al solo primo

elemento della lista.
Tale tempo è quindi trascurabile.

- $t_{pde}$: tempo necessario per prelevare tutte le dipendenze necessarie ad un controllo;trascurabile a causa della struttura scelta per la loro memorizzazione (hashtable).
- $t_{as}$: tempo necessario per applicare un selettore è trascurabile.
- $t_{inso}$: Rappresenta il tempo necessario a posizionare nella in modo ordinato un controllo per la sua prossima esecuzione.
  Questo tempo per come presentato non rappresenta una costante ma per comodità assumeremo che il numero di elementi da scorrere sia quello medio cioè $N_c/2$ con $N_c$ uguale al numero di controlli.

Ipotizzando di avere $N$ dipendenze $t_{p_1}, t_{p_2}, ..., t_{p_N}$ e con $t_c$ il il tempo necessario alla funzione di controllo, il tempo di esecuzione assicurabile per un controllo è:

$$[(t_{p_1} + t_{p_2} + ... + t_{p_N}) + t_c] \leq (10N + t_c)$$

Tale risultato è una sovrastima del risultato pratico reale. Esistono infatti una serie di contesti in cui tale stima viene notevolmente ribassata. Nel caso in cui una dipendenza sia presente in due controlli $C1$ e $C2$, con $t_{C1} < t_{C2}$, $C1$ non ha bisogno di spendere l'intero tempo della dipendenza. Inoltre ogni controllo applica una strategia di skip.
Tale strategia consiste nel risolvere altre dipendenze ogni volta che si incontra una di queste in stato ***EXEC***.

Si osserva inoltre come il multithreading migliori le prestazioni del sistema. Utilizzando più di un thread, è possibile far spendere il tempo di esecuzione di un poll ad un solo thread, con gli altri che avanzano nella risoluzione delle dipendenze.
In caso di timeout, i dieci secondi massimi teorici, vengo spesi nel ciclo di attesa solo se non si hanno altre dipendenze da risolvere.
Da sottolineare come sia auspicabile avere controlli che condividono molte dipendenze con intervalli di esecuzione vicini. In questo modo è possibile evitare di interrogare una componente troppo spesso, e sfruttare il multithreading in modo efficiente.
Si nota come,la struttura fino a qui descritta favorisce la definizione di controlli che operano sullo stesso insieme di dati.
Tutti le strategie viste viste, hanno infatti lo scopo di favorire le condivisione delle informazioni.

## 6.3 Problematiche Aperte

Nello sviluppo del software sono stati identificate alcune criticità.
Si elenca quali esse siano e le azioni da intraprendere per eliminarle o limitarle.

Un primo problema noto risiede nell'intervallo di esecuzione di un controllo. Definito infatti con $N$ il numero di dipendenze,$t_c$ il tempo per eseguire la funzione di controllo e con $T$ il tempo di esecuzione di un controllo.
Se $T < N * 10 + t_c$ è possibile che non tutte le esecuzioni siano eseguite. Si rende quindi necessario definire un tempo massimo per l'esecuzione di un controllo dato dall' intervallo di esecuzione.

Altro aspetto critico riguarda l'alterazione della coda dei controlli. Se infatti un thread controlla la coda ed osserva che il prossimo tempo di esecuzione del primo controllo è tra $T$ secondi, si pone in sleep per tale tempo.
Ipotizzando che a $T + 1$ secondi venga alterata la coda a causa di una certa dipendenza. In tal caso il sistema non riesce a mandare subito in esecuzione il thread in sleep per $T - 2$ secondi.

Un altro aspetto da migliorare riguarda il salvataggio degli stati. Sebbene sia importante salvare molti stati nei momenti critici, abbiamo notato che in molti casi si vanno a salvare una serie di stati contenenti le stesse informazioni di errore. Diviene fondamentale ottimizzare questo aspetto al fine di non avere un insieme troppo grande di stati di errore inutili.

Nel prossimo capitolo trarremo le conclusioni del lavoro svolto.
Ripercorreremo tutti gli aspetti analizzati nei vari capitoli per capire quali risultati siano stati ottenuti.
Identificheremo quali sono i punti da evolvere per migliorare l'architettura.

# 7 Conclusioni

Nell capitolo 2 si sono introdotte quali sono le ragioni che hanno motivato il lavoro di tesi. Oltre che per interesse personale, le ragioni che hanno interessato lo studio sono emerse da esperienze dirette con vari software di monitoring.
Lo sviluppo dei punti cardine hanno preso forma nel capitolo 3. In tale sezione, si sono stilati una serie di requisiti fondamentali per capire se esistono soluzioni che risolvono tali funzionalità.
Il passo successivo è quindi stato quello di analizzare lo stato dell'arte.
Si sono analizzati vari software di monitoring, confrontando le loro specifiche con i requisiti imposti in 3. I risultato finale è stato esposto nel paragrafo 4.5, in cui si sono identificate alcune mancanze oggettive. Tra queste si pongono sicuramente: la difficoltà a correlare metriche, la mancanza di poter avere tempi di interrogazione variabili e l'essere in grado valutare l'andamento di una componente.
Nel capitolo 5 si sono definiti concetti e idee alla base dell' architettura. Tra questi si hanno il formato dei dati, il concetto di diff, ecc. Nella sezione 6 si è passati a validare l'architettura, descrivendo le difficoltà implementative e valutandone le prestazioni.

Analizzando lo stato finale dei requisiti emerge che tra i risultati positivi ottenuti, assume posizione rilevante la correlazione di metriche; tale caratteristica giova dell'implementazione scelta per i controller.
L' ottimizzazione delle richieste ha assunto ruolo centrale grazie alla implementazione del concetto di stato e del concetto di selettore.
Inoltre, l'introduzione di un sistema per effettuare diff tra stati, ha permesso di valutare l'andamento di una componente.
Tra i requisiti che necessitano ulteriore approfondimento, rientra l'integrazione; questa potrebbe essere soddisfatta tramite lo sviluppo di un sistema di notifica modulare.
Come evidenziato nel paragrafo dedicato alle criticità note, la concorrenza necessita di ulteriore affinamento, in quanto non migliora come voluto le prestazioni.
Nel corso della validazione si è evidenziata la necessità di migliorare l'aspetto configurativo; ciò sarebbe possibile attraverso lo sviluppo di una interfaccia grafica.
Le carenze sopra descritte e le rispettive soluzioni, saranno prese in considerazione per successive implementazioni.

## Riferimenti bibliografici


[1] Deb Agarwal, Jose Maria Gonzalez, Guojun Jin e Brian Tierney. An infrastructure for passive network monitoring of application data streams. 2003. URL: http://www.escholarship.org/uc/item/66j721d5.

[2] A. Albano, G. Ghelli e R. Orsini. Fondamenti di basi di dati. Zanichelli, 2005. ISBN: 9788808170033. URL: https://books.google.it/books?id=pVPCAAAACAAJ.

[3] Tom Ammon. "Monitoring Concepts and Nagios Configuration Tutorial". Gen. 2014. URL: http://www.macos.utah.edu/documentation/administration/nagios/mainColumnParagraphs/010/document/2007.08.03-univ_of_utah-nagios.pdf.

[4] BMC. TrueSight Pulse API Documentation. URL: http://premium-documentation.boundary.com/metrics.

[5] BMC. TrueSight Pulse Boundary Plugin Shell. URL: https://help.boundary.com/hc/en-us/articles/201548962-Boundary-Plugin-Shell-Graph-anything-with-ease.

[6] L.W. Chu, S.H. Zou, S.D. Cheng, W.D. Wang e C.Q. Tian. "Internet service fault management using active probing in uncertain and noisy environment". In: 4 (ago. 2009), pp. 1–5. DOI: 10.1109/CHINACOM.2009.5339962.

[7] IAN D CIVIL e C WILLIAM SCHWAB. "The Abbreviated Injury Scale, 1985 revision: a condensed chart for clinical use." In: Journal of Trauma and Acute Care Surgery 28.1 (1988), pp. 87–90.

[8] Gerald Coley. "Beaglebone black system reference manual". In: Texas Instruments, Dallas (2013).

[9] Datadog. Datadog Agent Usage. URL: http://docs.datadoghq.com/guides/basic_agent_usage/.

[10] Datadog. Datadog Outlier Detection. URL: http://docs.datadoghq.com/guides/outliers/.

[11] Ethan Galstad. NDOUTILS Documentation Version 1.4. 6. Nagios Enterprise. 2007. URL: https://assets.nagios.com/downloads/nagioscore/docs/ndoutils/NDOUtils.pdf.

[12] C. Ghezzi, M. Jazayeri e D. Mandrioli. Ingegneria del software: fondamenti e principi. Pearson Education Italia, 2004. ISBN: 9788871922041. URL: https://books.google.it/books?id=0CM0celf2r8C.

[13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic e Marimuthu Palaniswami. “Internet of Things (IoT): A vision, architectural elements, and future directions”. In: Future Generation Computer Systems 29.7 (2013), pp. 1645–1660. ISSN: 0167-739X. DOI: http://dx.doi.org/10.1016/j.future.2013.01.010. URL: http://www.sciencedirect.com/science/article/pii/S0167739X13000241.

[14] David Lee, Arun N.Netravali, Krishan K. Sabnami, BInay Sugla e Ajita John. “Passive testing and applications to network management”. In: Bell Laboratories, Lucent Technologies 1 (1997).

[15] Bruce B. Lowekamp. “Combining Active and Passive Network Measurements to Build Scalable Monitoring Systems on the Grid”. In: SIGMETRICS Perform. Eval. Rev. 30.4 (mar. 2003), pp. 19–26. ISSN: 0163-5999. DOI: 10.1145/773056.773061. URL: http://doi.acm.org/10.1145/773056.773061.

[16] Lu Lu, Zhengguo Xu, Wenhai Wang e Youxian Sun. “A new fault detection method for computer networks”. In: Reliability Engineering & System Safety 114.3 (2013), pp. 45–51. ISSN: 0951-8320. DOI: http://dx.doi.org/10.1016/j.ress.2012.12.015. URL: http://www.sciencedirect.com/science/article/pii/S0951832013000045.

[17] R.E. Miller e K.A. Arisha. “Fault identification in networks by passive testing”. In: Simulation Symposium, 2001. Proceedings. 34th Annual. 2001, pp. 277–284. DOI: 10.1109/SIMSYM.2001.922142.

[18] Sophon Mongkolluksamee. “Strengths and limitations of Nagios as a network monitoring solution”. In: Proceedings of the 7th International Joint Conference on Computer Science and Software Engineering (JCSSE 2010). 7. 2009.

[19] Sophon Mongkolluksamee. “Strengths and limitations of Nagios as a network monitoring solution”. In: Proceedings of the 7th International Joint Conference on Computer Science and Software Engineering (JCSSE 2010). 2009, pp. 96–101.

[20] Stefan Ried, Holger Kisker e Pascal Matzke. “The evolution of cloud computing markets”. In: Forrester Research (2010).

[21] I. Sommerville. Ingegneria del software. Pearson, 2007. ISBN: 9788871923543. URL: https://books.google.it/books?id=h-hCKFMbqNMC.

[22] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall e G.J. Minden. “A survey of active network research”. In: Communications Magazine, IEEE 35.1 (1997), pp. 80–86. ISSN: 0163-6804. DOI: 10.1109/35.568214.

[23] Hasan Ural e Zhi Xu. "Testing of Software and Communicating Systems: 19th IFIP TC6/WG6.1 International Conference, TestCom 2007, 7th International Workshop, FATES 2007, Tallinn, Estonia, June 26-29, 2007. Proceedings". In: a cura di Alexandre Petrenko, Margus Veanes, Jan Tretmans e Wolfgang Grieskamp. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Cap. An EFSM-Based Passive Fault Detection Approach, pp. 335–350. ISBN: 978-3-540-73066-8. DOI: 10.1007/978-3-540-73066-8_23. URL: http://dx.doi.org/10.1007/978-3-540-73066-8_23.

[24] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres e Maik Lindner. "A Break in the Clouds: Towards a Cloud Definition". In: SIGCOMM Comput. Commun. Rev. 39.1 (dic. 2008), pp. 50–55. ISSN: 0146-4833. DOI: 10.1145/1496091.1496100. URL: http://doi.acm.org/10.1145/1496091.1496100.

[25] Wikipedia. Reti Bayesiane — Wikipedia, The Free Encyclopedia. [Online; accessed 7-September-2015]. 2014. URL: https://it.wikipedia.org/wiki/Reti_Bayesiane.

[26] Zabbix 2.4 Documentation. 2015. URL: https://www.zabbix.com/documentation/2.4/manual/concepts/definitions.

[27] Zabbix 2.4 Trigger Expression Documentation. 2016. URL: https://www.zabbix.com/documentation/2.0/manual/config/triggers/expression.