Analisi e Descrizione Iniziali del Plugin
CheckTime V0.3

Tommaso Fontana

17/07/2019

Abstract

Descrizione iniziale del plugin che analizzando i valori di utilizzo del disco di una macchina di un dato periodo di tempo Δt stima quanto tempo rimane prima della saturazione di un risorsa e genera warning nel caso il tempo sia inferiore a certe threshold. L'utilizzo principale pensato per lo script e' monitorare lo stato dei dischi ma lo script e' abbastanza generale da poter trattare anche altri casi (e.g. CPU/RAM) purche' sia definito il valore attuale e quello massimo.

Descrizione del problema

Attualmente le threshold di warning e critical per i dischi dei server sono espresse in bytes.

Questo ovviamente risulta poco utile nel caso di server maturi il cui utilizzo del disco cambia poco nel tempo.

Ad esempio un server il cui unico scopo e' agire da firewall avra' raramente bisogno di occupare ulteriore spazio su disco, quindi anche nel caso avesse libero solo 1Gb di spazio sul disco questo potrebbe funzionare indisturbato per anni.

L'esempio diametralmente opposto e' un server che ospita un Database il quale potrebbe occupare quel Gigabyte in una manciata di ore.

Quindi chiaramente lo spazio rimanente non e' una metrica affidabile ed utile in generale.

La soluzione proposta da questo plugin e' quella di stimare quanto tempo rimane alla saturazione del disco (raggiungimento del 100% di utilizzo) analizzando lo storico dati dell'ultimo periodo.

Il valore aggiunto di potere, a questo punto, avere le threshold espresse in una qualche unita' di tempo e' la possibilita' di essere avvisati tempestivamente.

Ad esempio che supponendo che per ordinare un disco aggiuntivo ed installarlo servano al massimo 5 giorni si poter impostare la threshold critica a 6 giorni in modo da essere avvisati un giorno prima del tempo minimo per garantire la continuità del servizio.

E si potrebbe impostare la threshold di warning ad esempio ad 1 mese in modo da poter ordinare i dischi in anticipo in modo da essere preparati al' eventuale saturazione del disco.

Inoltre cio' evita falsi allarmi, nel caso del firewall enunciato prima l'utente avrebbe un messaggi di warning / critical anche se potrebbero volerci anni a saturare il disco effettivamente.

Infine permette di avere dati oggettivi e quantificabili su cui poter ragionare invece che grafici che lasciano spazio all'interpretazione. Questo permette un'organizzazione piu' precisa ed oggettiva.

Il caso dei dischi e' solo uno dei vari casi in cui avere una metrica espressa in tempo risulta utile rispetto ad averla espressa nella sua unita'.

Un altro caso potrebbe essere l'utilizzo di RAM / CPU di un server.
Con il passare del tempo, ci si aspetta che il numero di utenti cresca.
Con piu' utenti attivi il servizio richiedera' piu' risorse, quindi con questo script potremmo prevedere quando servira' fare scaling (verticale o orizzontale) dei server che gestiscono il serivzio.
Il principale vantaggio e' che questo permette di creare una stima di un piano di investimento nel tempo per adeguare l'hardware alla crescita del servizio.

Descrizione Implementazione del Plugin

Premessa: Come verra' descirtto piu' avanti avremo bisogno di due frontend, uno per windows ed uno per linux. Questi sono *checktimewin, checktimelinux* ma in generale quando non e' necessario specificare il sistema operativo perche' si parla di feature comuni verra' scritto solamente *checktime* omettendo l'OS per quanto l'eseguibile *checktime* non esiste.

Descrizione Installazione e Configurazione

Lo script non avra' bisogno di alcuna dipendenza poiche' autocontenuto nella cartella.
La procedura di installazione sara' soltanto la copia della cartella sul sistema e la creazione del file di cron-job che avra' sintassi simile a:

```
*/5 * * * * /var/checktime/checktime -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace" -m "Win" -n "1d" -w "4w" -c "1w"
```

Aggiunta dello script al PATH

Si puo' creare un link simbolico:

```
$ sudo ln -s /var/checktime/checktimewin   /bin/checktimewin
$ sudo ln -s /var/checktime/checktimelinux /bin/checktimelinux
```

cosi che in tutto il sistema si possa chiamare *checktime* come qualsiasi altro commando nel *PATH*.

Cio' permette di avere il file di cron che non deve avere hardcodato il path di installazione di *checktime*:

```
*/5 * * * * checktime -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace" -m "Win" -n "1d" -w "4w" -c "1w"
```

Configurazione accesso DB

Nella cartella ci sara' un file *db_settings.json* che conterra' le settings per connettersi al DB che avra' sintassi simile a:

```
{
    "database": "icinga2",
    "host": "127.0.0.1",
    "port": 8086,
    "username": "root",
    "password": "",
    "ssl": true,
    "verify_ssl": true,
    "timeout": 60,
    "retries": 3,
    "use_udp": false,
    "udp_port": 4444,
    "proxies": {},
    "path": ""
  }
```

Descrizione Aggiornamento

Il plugin verrà sviluppato in un repository quindi una volta rilasciata la nuova versione bastera' eseguire il seguente comando (o comunque una sua versione alternativa) nella cartella di installazione per aggiornare:

```
$ git pull
```

Descrizione Utilizzo

Input

Poiche' gli schema dei db sono diversi a seconda del fatto che il sistema sia Windows o Linux avremo bisogno di due "frontend" per poter passare i dati allo script core.
Rispettivamente gli eseguibili *checktimewin* e *checktimelinux*.
Quando possibile, per essere il piu' generali possibili verra' scritto *checktime* omettendo il sistema operativo specifico per quanto l'eseguibile *checktime* non esiste.

Nello specifico gli schema dei measurements per i due sistemi sono:

```
Windows:
time, hostname, matric, service, warn, crit, max, unit, value
Linux:
time, device, free, fstype, host, inodes_free, inodes_total,
    mode, path, total, used_percent
```

Windows frontend

```
$ ./checktimewin -h
usage: main_win.py [-h] -M MEASUREMENT -n WINDOW -w WARNING_THRESHOLD -c
                   CRITICAL_THRESHOLD [-v {0,1}]-H HOST -s SERVICE
                   [-m METRIC] [-e EXCLUDE]

optional arguments:
  -h, --help            show this help message and exit

query settings (required):
  -M MEASUREMENT, --measurement MEASUREMENT
                        measurement where the data will be queried.

thresholds settings:
  -n WINDOW, --window WINDOW
                        the range of time to consider in the analysis.
  -w WARNING_THRESHOLD, --warning-threshold WARNING_THRESHOLD
                        the time that if the predicted time is lower the
                        script will exit(1).
  -c CRITICAL_THRESHOLD, --critical-threshold CRITICAL_THRESHOLD
                        the time that if the predicted time is lower the
                        script will exit(2).

verbosity settings (optional):
  -v {0,1}, --verbosity {0,1}
                        set the logging verbosity, 0 == ERROR, 1 == DEBUG, it
                        defaults to ERROR.

os dependant settings (required) Windows:
  -H HOST, --host HOST host which disks will be checked.
  -s SERVICE, --service SERVICE
                        service to be checked.

os dependant settings (optional) Windows:
  -m METRIC, --metric METRIC
                        metric to be checked.
  -e EXCLUDE, --exclude EXCLUDE
                        metric to be excluded from the analysis.
```

Esempio di utilizzo

```
$ checktime  -M "disk"  -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace" -m "C:/" -n "1d" -w "4w" -c "1w"
```

Il quale controllera' dal measurement *disk* per l'host *"rt-sccm01-p1.idolrt.regione.toscana.it"*, il servizio *"Diskspace"* per la metrica *"C:/"* utilizzando i dati dell'ultimo giorno e generera' un warning se il tempo previsto e' inferiore a 4 settimane ed un critical error nel caso sia inferiore ad una settimana.

Linux frontend

Lo script dovra' sapere su quale Host, Servizio e opzionalemnte Metrica fare la analisi. Questi dati dovranno essere passati allo script tramite riga di comando nella seguente sintassi:

```
$ ./checktimelinux -h
usage: main_linux.py [-h] -M MEASUREMENT -n WINDOW -w WARNING_THRESHOLD -c
                     CRITICAL_THRESHOLD [-v {0,1}] -H HOST -d DEVICE
                     [-p PATH] [-e EXCLUDE]
optional arguments:
  -h, --help            show this help message and exit

query settings (required):
  -M MEASUREMENT, --measurement MEASUREMENT
                        measurement where the data will be queried.

thresholds settings:
  -n WINDOW, --window WINDOW
                        the range of time to consider in the analysis.
  -w WARNING_THRESHOLD, --warning-threshold WARNING_THRESHOLD
                        the time that if the predicted time is lower the
                        script will exit(1).
  -c CRITICAL_THRESHOLD, --critical-threshold CRITICAL_THRESHOLD
                        the time that if the predicted time is lower the
                        script will exit(2).

verbosity settings (optional):
  -v {0,1}, --verbosity {0,1}
                        set the logging verbosity, 0 == ERROR, 1 == DEBUG, it
                        defaults to ERROR.

os dependant settings (required) Linux:
  -H HOST, --host HOST host which disks will be checked.
  -d DEVICE, --device DEVICE
                        service to be checked.

os dependant settings (optional) Linux:
  -p PATH, --path PATH metric to be checked.
  -e EXCLUDE, --exclude EXCLUDE
                        path to be excluded from the analysis.
```

Cornercases

Poiche' si cerca di mantenere il comportamento dei frontend il piu' omogeneo possibile varra' la equivalenza tra:

```
-d DEVICE <-> -s SERVICE
-p PATH   <-> -m METRIC
```

e gli *-e* funzioneranno esattamente nello stesso modo pero' si riferiranno a *PATH* o a *METRIC* a seconda del frontend.

La gerarchia dei field per i due sistemi quindi sara':

```
Windows:
Measurement > Host > Service > Metric
Linux:
Measurement > Host > Device > Path
```

Per cercare di rendere la trattazione il piu' omogenea e predittbile possibile.

Cornercase: Cosa succede se uno o piu' parametri richiesti non sono passati

Se un parametro rischiesto o piu' non viene passato lo script uscira' con **exit(1)** ed elenchera' quali parametri mancano sulla **stderr**.
e.g. se lo script viene chiamato senza parametri:

```
$ checktime
usage: checktime [-h] -M MEASUREMENT -H HOST -s SERVICE [-m METRIC] [-e EXCLUDE]
                 -n WINDOW -w WARNING_THRESHOLD -c CRITICAL_THRESHOLD [-v {0,1}]
checktime: error: the following arguments are required:
-M/--measurement, -H/--host, -s/--service, -n/--window,
-w/--warning-threshold, -c/--critical-threshold
```

Cornercase: Cosa succede se uno o piu' parametri non esisto sul DB

Se uno o piu' parametri non esistono sul DB (eccetto le metriche) allora lo script uscira' con **exit(1)** e descrivera' quali parametri non sono stati trovati sulla **stderr**.
Lo script si ferma al primo parametro non trovato con ordine di priorita':

```
Windows:
Measurement > Host > Service > Metric
Linux:
Measurement > Host > Device > Path
```

Il caso in cui una metrica non esiste viene affrontato piu' avanti.
e.g. se si sbaglia a scrivere *"disk"* e si scrive *"disck"*:

```
$ checktime -M "disck" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace" -n "1d" -w "4w" -c "1w"
error:  measurement "disck" not found.
```

e.g. se si sbaglia a scrivere l'host:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.com"
    -s "Diskspace" -n "1d" -w "4w" -c "1w"
error:  host "rt-sccm01-p1.idolrt.regione.toscana.com" not found.
```

e.g. se si sbaglia a scrivere il service:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "diskspace" -n "1d" -w "4w" -c "1w"
error:  service "diskspace" not found.
```

Scelta delle metriche / path

Selezione delle metriche / path

Se allo script non viene passato nessun parametro *-m/-p* allora lo script selezionera' di default tutte le metriche / path presenti sul DB.

e.g.:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace" -n "1d" -w "4w" -c "1w"
/ 1y3d4s
/produzione 3w
/var 5d
/mnt 6w
/tmp 1s
```

Se invece vogliamo analizzare solo un subset ti metriche queste dovranno essere passate ciascuna con un parametro *-m*.

e.g.:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace" -p "/" -p "/produzione" -n "1d" -w "4w" -c "1w"
/ 1y3d4s
/produzione 3w
```

Filtraggio delle metriche / path

Se si vuole escludere qualche metrica / path dalla analisi basta passarla al parametro *-e*.

e.g. analizzare tutto ma escludere */tmp, /var, /mnt*:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace"  -e "/tmp" -e "/var" -e "/mnt" -n "1d" -w "4w" -c "1w"
/home 1d2s
/produzione 1y3d
```

Cornercase: presenza di entrambi *-m, -e*

I parametri *-m/p* ed *-e* sono combinabili.

e.g. questo script analizzera' solo */tmp*:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace"  -p "/tmp" -p "/var" -e "/var" -n "1d" -w "4w" -c "1w"
/tmp 1y2w4s
```

Questa scelta e' stata fatta per mantenere consistenza nel utilizzo dei parametri e perche' potrebbe essere utile in caso di debug di poter disabilitare una metrica temporaneamente senza il rischio di dimenticare quale fosse.

Cornercase: *-e* di una metrica non esistente

Nel caso eseguissimo un *-e* su una metrica non presente sul DB questa verrà ignorata e non genererà alcun tipo di errore o warning.

e.g. questo script non genera' errori o warning a causa della non esistenza di *C:/* o */tmp*:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace"  -e "C:/" -e "/tmp" -n "1d" -w "4w" -c "1w"
```

Cornercase: *-m/p* di una metrica / path non esistente

Nel caso eseguissimo un *-m/p* su una metrica / path non presente sul DB questa verrà ignorata nella analisi ma ritornerà uno stato di un warning (**exit(1)**) e vera' scritto un messaggio di warning sulla **stderr**.
(Nel caso lo script dovesse ritornare **exit(2)** a causa di un valore fuori threshold allora lo script darà priorità alla **exit(2)** rispetto al **exit(1)**)
e.g. questo script genera' un warning a causa della non esistenza di *C:/* o */tmp*:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace"  -m "C:/" -m "/tmp" -n "1d" -w "4w" -c "1w"
warning: Not found the metric "C:/"
/tmp 1w2d3m
```

0.1 Output

Nel caso lo script sia andato su una singola metrica l'output scritto sulla **stdout** sara' simile a:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace" -m "C:/" -n "1d" -w "4w" -c "1w"
C:/ 1w2d3h
```

Nel caso vada su mutiple metriche l'output sara' simile a;

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace" -n "1d" -w "4w" -c "1w"
C:/ 1w2d3h
D:/ 1y3d
E:/ 1m2s
F:/ 2y2w
```

In generale seguira' il linguaggio regolare:

```
(<Nome Metrica> <Tempo Rimasto>\n)+
```

Se il tempo rimasto di qualunque metrica sia inferiore al livello di warning lo script avra' exit code 1.
Nel caso qualunque metrica sia inferiore al livello di Critical lo script avra' exit code 2 altrimenti sara' 0.

Caso di tempo infinito

Nel caso che un disco abbia spazio utlizzato costante o decrescente l'output sara' del simile a:

```
$ checktime -M "disk" -H "rt-sccm01-p1.idolrt.regione.toscana.it"
    -s "Diskspace" -n "1d" -w "4w" -c "1w"
C:/  inf
D:/ 1y3d
E:/  inf
F:/ 2y2w
```

Formato del tempo

La sintassi del tempo dovra' seguire la seguente espressione regolare:

```
(inf|(\d+y)?(\d+w)?(\d+d)?(\d+h)?(\d+m)?(\d+s)?)
```

dove la stringa vuota sara' interpretata come 0 secondi.
di seguito esempi di tempo espressi in questa forma, uno per riga:

```
inf
1y
2000s
1y1w1d1h1m1s
1y10w100d1000h10000m100000s
1y3s
```

Descrizione Cartella

La cartella dello script avra' la seguente struttura:

```
drwxr-xr-x 5 user user 4,0K 29 giu 14.25 check_time_venv
drwxr-xr-x 4 user user 4,0K 29 giu 14.21 doc
drwxr-xr-x 3 user user 4,0K  2 lug 23.01 src
drwxr-xr-x 2 user user 4,0K 29 giu 13.51 tests
-rw-r--r-- 1 user user  281 29 giu 13.33 db_settings.json
-rw-r--r-- 1 user user  18K 29 giu 14.22 license
-rw-r--r-- 1 user user   95 29 giu 14.24 README.rst
-rwxr-xr-x 1 user user  102 29 giu 16.26 checktimewin
-rwxr-xr-x 1 user user  103 29 giu 16.27 checktimelinux
```

Legenda:

- **check_time_venv** E' la cartella del virtual environment di python 3 gia' configurato.
- **doc** La cartella contenete la documentazione e gli esempi di utilizzo dello script.
- **src** La cartella con i sorgenti python dello script.
- **tests** La cartella conetenente i test da poter eseguire post installazione per poter testare che lo script funzioni correttamente.
- **db_settings.json** E' il file di configurazione per configurare come connettersi al DB.
- **license** La licenza GPL2 dello script.
- **README.rst** File di intro della repo e tutorial veloce su come installare, aggiornare ed utilizzare lo script.
- **checktimelinux** E' l'eseguibile per Linux.
- **checktimewin** E' l'eseguibile per Windows.

Descrizione del metodo di predizione

0.2 Descrizione del problema

Data una serie di valori $p_i = (t_i, u_i)$ su un intervallo di tempo Δt dove $u \in [0, 1] \subset \mathbb{R}$ e' l'utilizzo percentuale del disco e $t \in \mathbb{N}$ e' il timestamp della misurazione di u vogliamo sapere il tempo rimanente Δt_r al quale ci aspettiamo il raggiungimento massimo (100%) del disco.

0.3 Soluzione proposta

Per semplicità inizialmente la strategia e' quella di supporre una relazione lineare tra uso del disco (u) e il tempo (t):

$$u = mt + q$$

per ottenere i due parametri $m, q \in \mathbb{R}$ ci sono diverse metodologie spiegate dopo.
Una volta ottenuti i parametri m, q basta invertire l'equazione:

$$u = mt + q \quad \Rightarrow \quad t = \frac{u - q}{m}$$

quindi basta sostituire ad u l'utilizzo che per cui vogliamo stimare il tempo, in questo caso il 100% quindi :

$$t_r = \frac{1 - q}{m}$$

in fine per ottenere il tempo rimanente al istante t_{now} basta fare:

$$\Delta t_r = t_r - t_{now}$$

0.4 Soluzione proposta aggiuntiva nel caso di dati molto rumorosi

Nel caso i dati abbiano molte periodicita' o rumore si potrebbe pensare di eseguire prima di tutto uno smoothing dei dati con una moving window o con un kernel gaussiano.
Poi si potrebbe usare il metodo di *Holt Winter* quindi un *Double* o *Triple Exponential Smoothing* per estrarre trend e periodicita' (questo metodo e' intensivo computazionalmente quindi conviene usarlo solo se strettamente necessario).
Alternativamente al metodo di *Holt Winter* si potrebbe eseguire una *Trasformata Discreta di Fourier* della differenza tra i valori e il loro trend per identificare le componenti principali ed ad esempio tenere le piu' grandi per predirre i fenomeni periodici principali.
Questo permetterebbe anche al utente nel caso di avere degli insights sul proprio sistema.

0.5 Metodi di regressione

Per stimare la retta che approssima i dati ci sono vari modi, spiegati brevemente qui di seguito in ordine di complessità ed utilità.

Riassunti:

- Regressione dei minimi quadrati: Il metodo standard.
- RANSAC: Cerca di distinguere i valori dal rumore.
- Regressione Bayesiana: Possiamo suggerire al metodo come ci aspettiamo i risultati e sopratutto **possiamo stimare quanto e' probabile che tra un certo quantitativo di tempo il disco sara' pieno.**

0.5.1 Regressione minimi quadrati

E' il metodo piu' facile e veloce.
Definiamo una funzione che calcola l'errore in funzione dei due paramatri cercati m, q.

$$E(m,q) = \sum_{i=0}^{N} [u_i - (mt_i + q)]^2$$

E vogliamo i parametri che minimizzano l'errore quindi basta imporne il gradiente a zero.

$$\nabla E(m,q) = 0$$

Il che implica che:

$$\frac{\partial E(m,q)}{\partial q} = 2\sum_{i=0}^{N} [u_i - (mt_i + q)] = 0 \rightarrow \sum_{i=0}^{N} u_i - m\sum_{i=0}^{N} t_i + qN = 0$$

$$\frac{\partial E(m,q)}{\partial m} = 2\sum_{i=0}^{N} [u_i - (mt_i + q)]\,[u_i] = 0 \rightarrow \sum_{i=0}^{N} u_i^2 - m\sum_{i=0}^{N} t_i u_i + q\sum_{i=0}^{N} u_i = 0$$

Le quali formano un sistema lineare di equazioni nelle due incognite m, q che una volta risolto si ottengono gli stimatori:

$$m = \frac{\sum_{i=0}^{N} \left[t_i - \hat{t}\right]\left[u_i - \hat{u}\right]}{\sum_{i=0}^{N} \left[t_i - \hat{t}\right]^2}$$

$$q = \hat{u} - m\hat{t}$$

dove $\hat{t}, \hat{u}$ sono rispettivamente i valori medi di tempo ed utilizzo nella finestra temporale considerata.

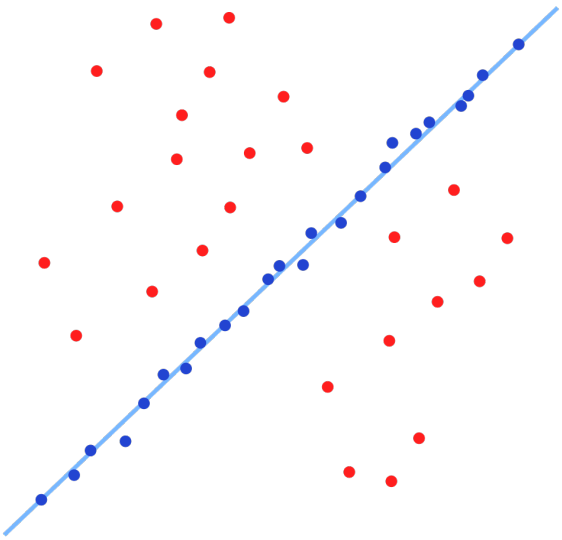
0.5.2 RANSAC (RANdom SAmple Consensus)

E' un metodo robusto per fare fitting di dati anche in presenza di outliers quindi rumore.

È un algoritmo non deterministico nel senso che produce un risultato corretto solo con una data probabilità, che aumenta al crescere delle iterazioni consentite.

L'assunzione di base per il funzionamento è che i dati siano costituiti da dei dati giusti e da dei dati di rumore.

Un esempio di applicazione in cui i punti blu sono ritenuti dati e quelli rossi vengono classificati come rumore:



0.5.3 Regressione Bayesiana

Lo scopo della regressione bayesiana non e' di stimare puntualmente i parametri m, q ma stimarne la distribuzione.

Ciò ha tre Vantaggi.

1. Possiamo scegliere con che sicurezza (probabilita') vogliamo la stima.
2. SI puo' avere degli intervalli di confidenza, quindi un modo oggettivo per misurare quanto la stima sia affidabile.
3. Questo tipo di regressione parte da una distribuzione detta *prior* che viene modificata dai dati. Questa distribuzione possiamo sceglierla per "suggerire" il comportamento che ci aspettiamo, cioè che ad esempio l'utilizzo nel tempo tenderà a salire o rimanere uguale e raramente scenderà.

E' ragionevole assumere che i dati di utilizzo abbiamo un errore gaussiano additivo, quindi e' possibile rappresentare l'utilizzo del disco come:

$$u_i = r_i + \mathcal{N}(0, \sigma^2)$$

dove r_i rappresenta il reale utilizzo al istante i.
Questo implica che anche l'utilizzo sia rappresentabile da una distribuzione normale:

$$u_i \sim \mathcal{N}(r_i, \sigma^2)$$

Ora usando il teorema di Bayes possiamo calcolare la distribuzione dei parametri:

$$P(m|U,T) = \frac{P(U|m,T)P(m|T)}{P(U|X)}$$

dove $P(m|U,T)$ e' la distribuzione di probabilità' del parametro m sapendo i dati di utilizzo e quelli di tempo U, T.

e più importante e' $P(m|T)$ e il *prior* che per *"""non aggiungere informazione"""* potremmo considerare uniforme ma nel nostro caso per favorire i valori di pendenza positivi e sopratutto rendere più probabili i valori piccoli rispetto ad quelli grandi poiché lo spazio occupato salirà lentamente possiamo usare un *prior* esponenziale.

Questo metodo non e' applicabile direttamente su variabili continue ma possiamo approssimarlo usando un metodo MCMC nello specifico credo che il Campionamento di Gibbs sia ottimale.

Esempio di stima bayesiana (la distribuzione a campana) vs stima puntuale (il singolo punto rappresentato dalla riga rossa tratteggiata).