Matteo Baccan

Succo di lampone: come ottimizzare JAVA e PHP su un'architettura Raspberry Pi Cluster

matteo@baccan.it – http://www.baccan.it

Chi sono?

Matteo Baccan

Divulgatore informatico, giornalista per riviste tecniche nazionali oltre 700 articoli per: Dev, Login, Computer Programming, Mokabyte. Relatore tecnico ad eventi di programmazione: Borland Forum 2000, Webbit 2004, JIP day 2005, Javaday (2006,2007,2010) PHPDay (2008, 2010), CONFSL 2010, WebTech 2010, Codemotin (2011,2012,2013,2014). Autore di Corso di C# ISBN 8881500167. Autore di JobCrawler e HTML2POP3 (Oltre 900.000 download su SourceForge)

Il mio motto

Per fare un grande piatto devi togliere non aggiungere

Gualtiero Marchesi
(a volte vale anche per un grande software)

Cosa vedremo

- Cos'è Raspberry Pi
- Cosa ci possiamo fare
- Alcune ottimizzazioni del sistema di base
- PHP su Raspberry Pi
- Java su Raspberry Pi
- Alternative per lo sviluppo
- Raspberry Pi Emulato

Non stiamo parlando dei lamponi

Cos'è Raspberry Pi?

Tratto da : http://it.wikipedia.org/wiki/Raspberry_Pi

Il Raspberry Pi è un **single-board computer** (SBC) sviluppato nel Regno Unito dalla Raspberry Pi Foundation.

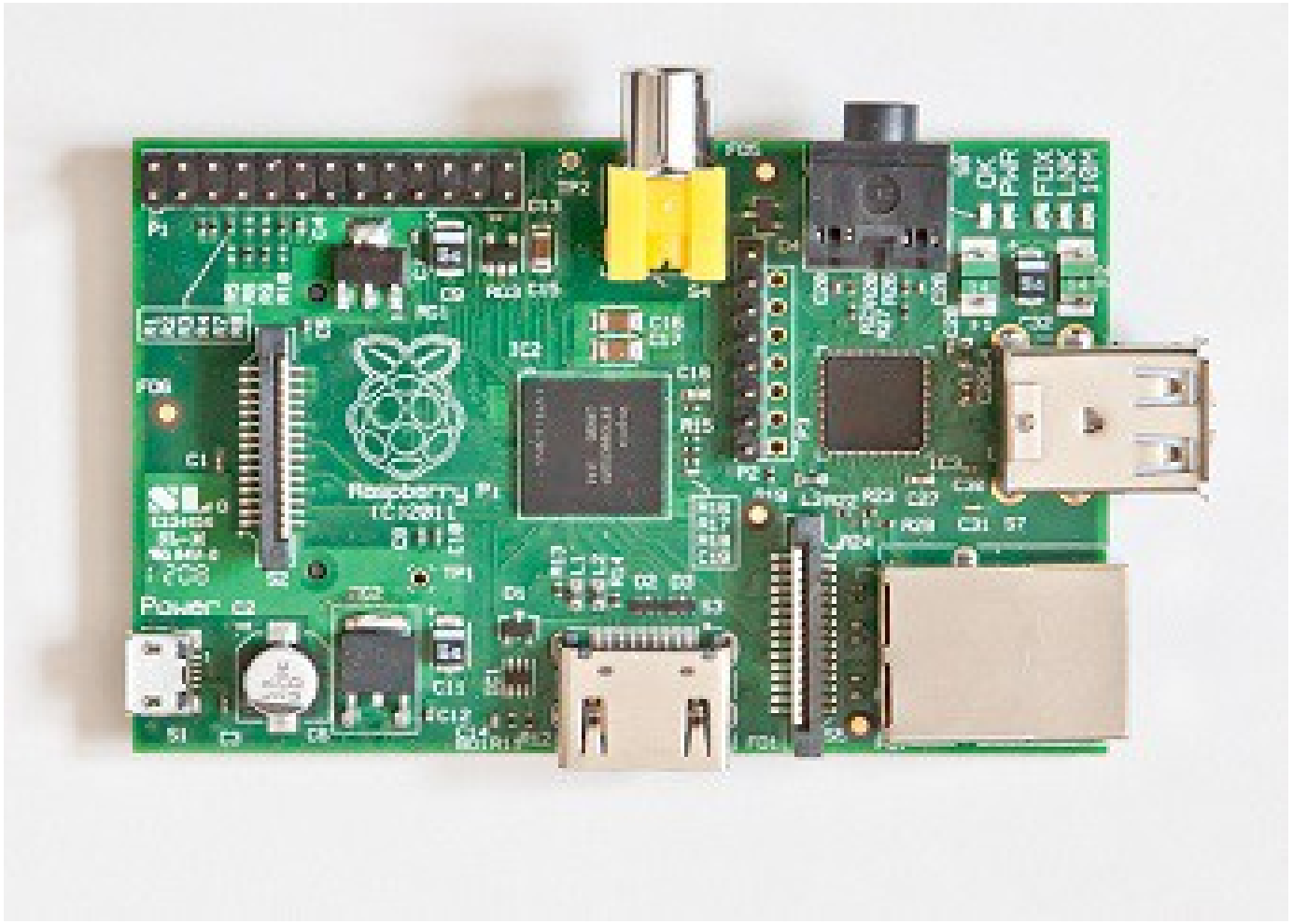
Il suo lancio al pubblico è avvenuto alla fine del mese di febbraio 2012.
L'idea di base è la realizzazione di un dispositivo economico, concepito per stimolare l'insegnamento di base dell'informatica e della programmazione nelle scuole

Il progetto ruota attorno a un System-on-a-chip (SoC) Broadcom BCM2835, che incorpora un processore **ARM1176JZF-S** a 700 MHz, una GPU VideoCore IV, e 256 o 512 Megabyte di memoria.

Il progetto non prevede né hard disk né una unità a stato solido, affidandosi invece a una scheda SD per il boot e per la memoria non volatile.

La scheda è stata progettata per ospitare sistemi operativi basati su un kernel Linux o RISC OS.

Cosa comperate adesso



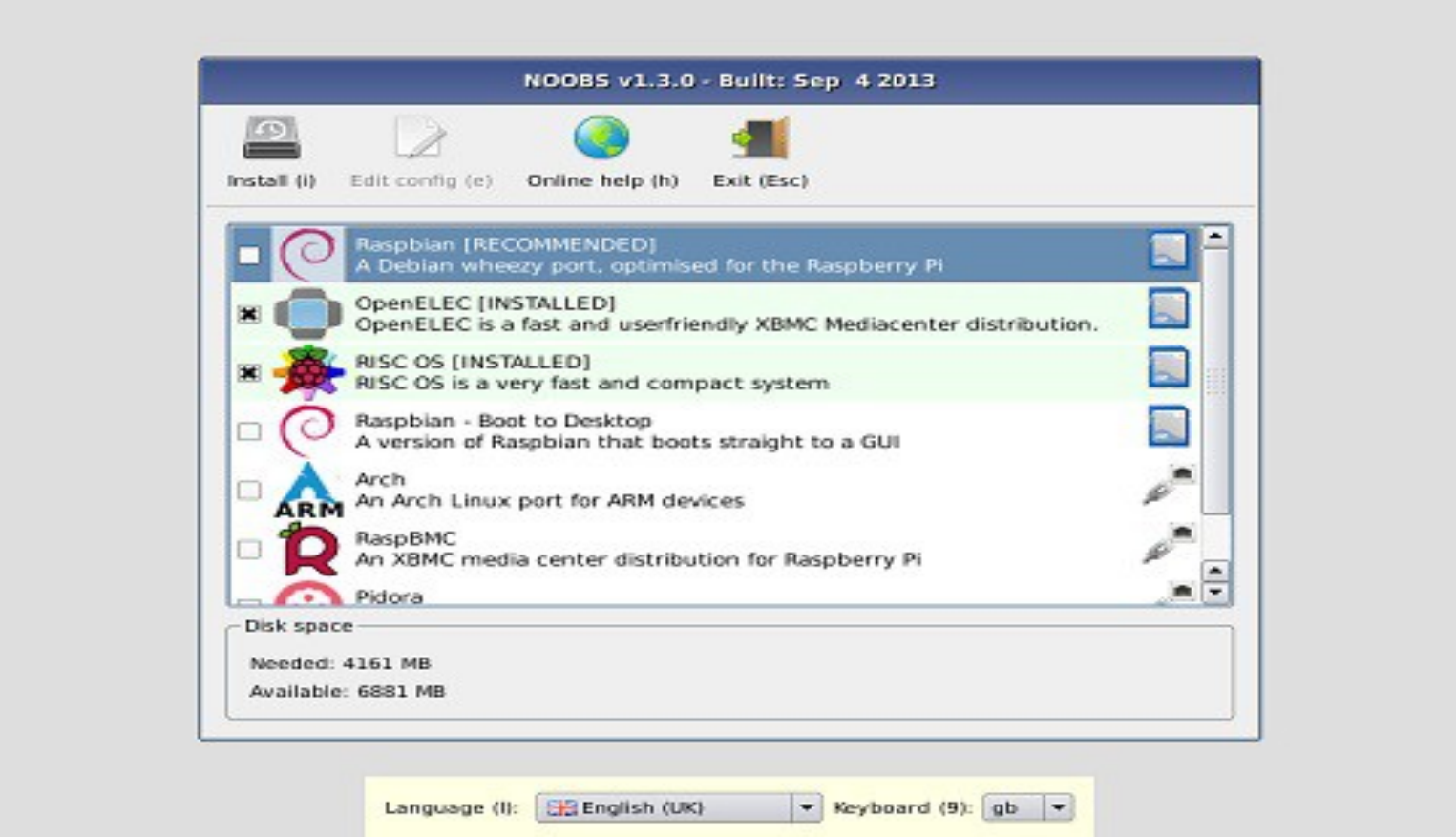
Cosa comprerete prossimamente

Raspberry Pi Compute Module

BCM2835 processor, 512Mbyte of RAM e 4Gbyte eMMC Flash device

Cosa installare

Il punto di partenza è sicuramente NOOBS (attualmente v 1.3.4)
`http://www.raspberrypi.org/downloads`



Cosa installare

All'interno di NOOBS (New Out Of Box Software) è possibile trovare una serie di sistemi operativi installabili

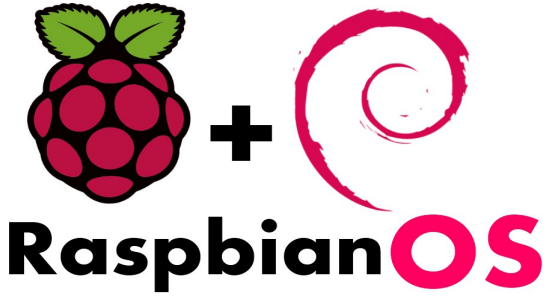
Raspbian : Il porting di Debian 7 "**Wheezy**", ottimizzato per Raspberry Pi

Alternative

Pidora: Pidora is a Fedora Remix optimized for the Raspberry Pi
Arch Linux: An Arch Linux port for ARM devices
OpenELEC: is a fast and userfriendly XBMC Mediacenter distribution.
RaspBMC: An XBMC media center distribution for Raspberry Pi
RISC OS: is a very fast and compact system

Nei nostri test useremo **Raspbian** basato su Debian 8 "**Jessie**"

Oggi parleremo di **Raspbian**



È una distribuzione pensata per essere “general purpose”, con un utilizzo **client**

Non esiste una versione **server** ufficiale di Raspbian

Vediamo insieme come costruirla.

Iniziamo a **spremere** il nostro lampone

Qualcuno si è posto il problema di non avere una distribuzione ottimizzata per una configurazione **server** e ha realizzato

Raspbian Server Edition 2.5
http://sirlagz.net/2013/07/19/raspbian-server-edition-2-5/

In realtà è una Raspbian basata su Debian 7 Wheezy portata a Debian 8 Jessie, con la rimozione di una serie di pacchetti "inutili" a livello server.

Parte dal concetto che: un "purista" considera **peccaminoso** l'utilizzo di una GUI su un server, viene quindi rimosso LXDE e tutti i relativi pacchetti, più una serie di pacchetti non necessari su un server

In alternativa potete provare

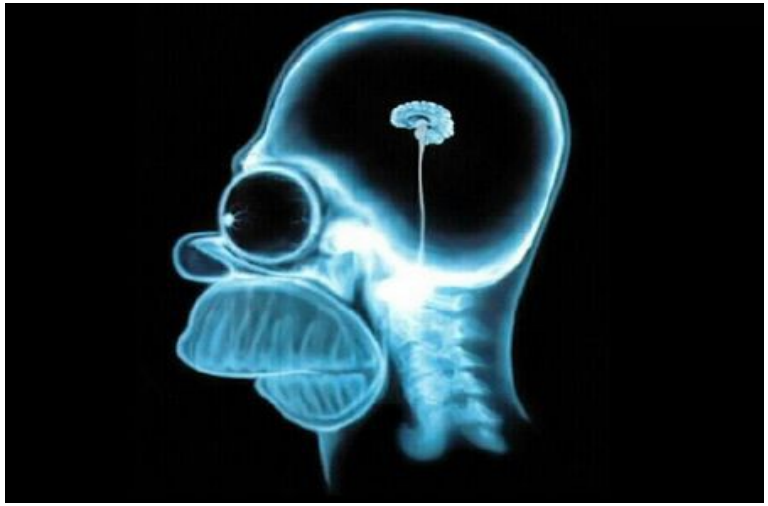
Minimal Raspbian – Server and XFCE editions
http://xecdesign.com/minimal-raspbian-server-xfce-editions/

Non dobbiamo però essere ottimisti: la sola rimozione dei pacchetti inutili **non basta** a realizzare una versione **server** di Raspberry Pi.

Vediamo quali altre tecniche ci permettono di migliorare i 2 aspetti che possono cambiare notevolmente le prestazioni di un Raspberry Pi

Velocità

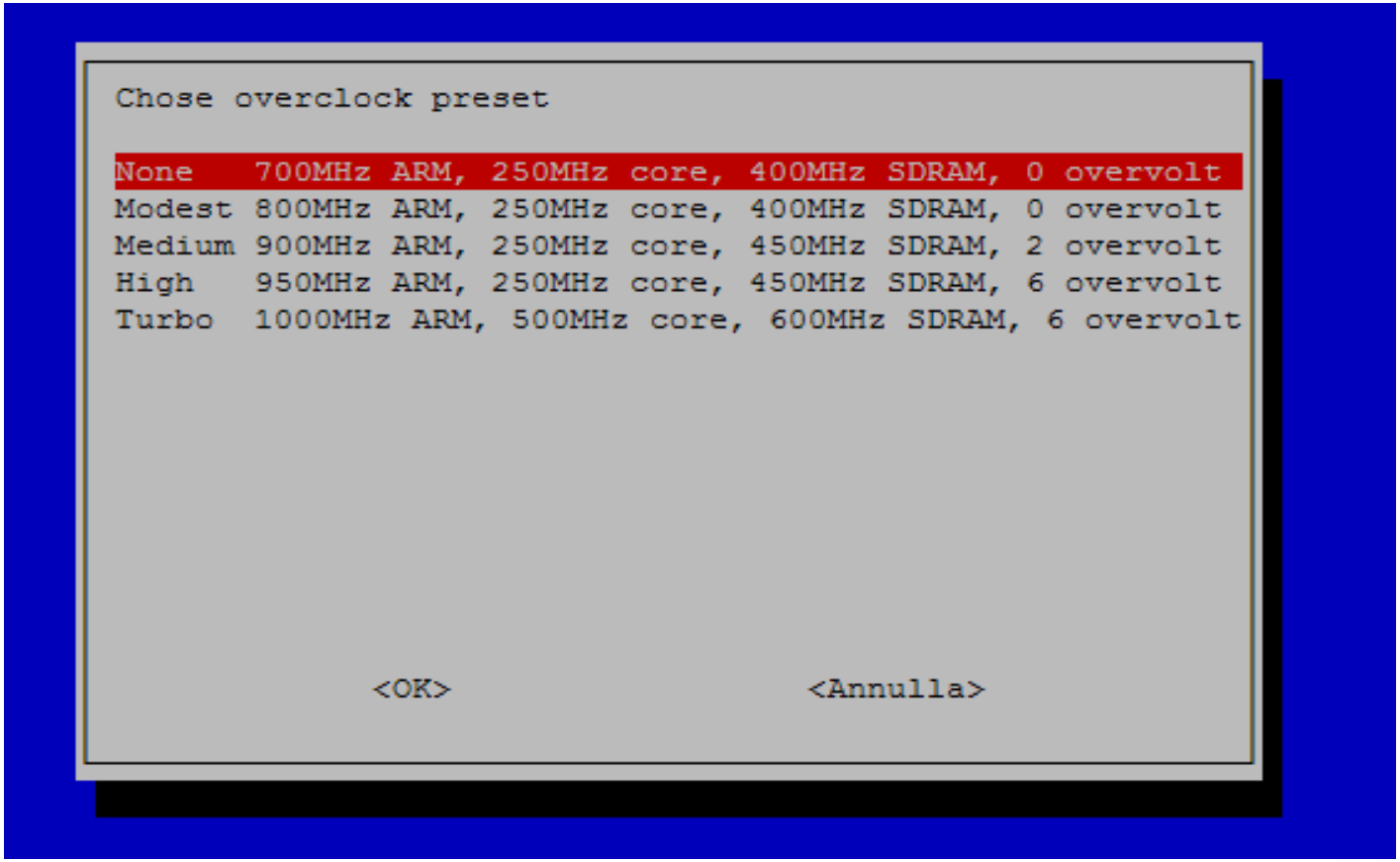
Memoria



Velocità: Overclock guidato

Chi non ha mai provato ad “overcloccare” il proprio computer?
Con Raspberry Pi l’overclock è ammesso e di sistema.

Tramite il tool di configurazione : sudo raspi-config



Velocità: Overclock manuale

Per chi volesse provare manualmente, basta modificare direttamente il file

```
sudo nano /boot/config.txt
```

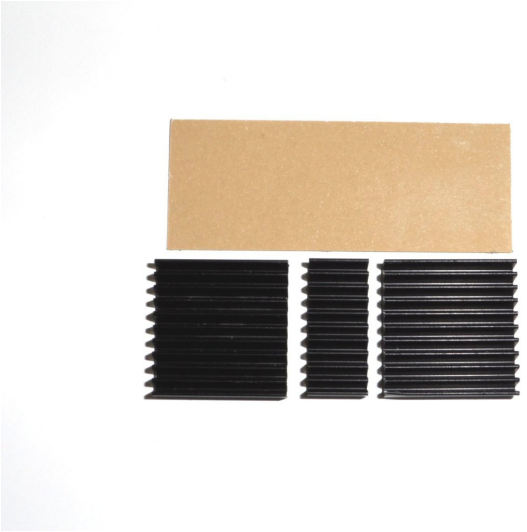
Sul forum Raspberry Pi sono indicate altre configurazioni possibili, che possono farci arrivare alla frequenza di 1150MHz.
Per info `http://elinux.org/RPiconfig`

arm_freq	gpu_freq	core_freq	sdram_freq	over_voltage
800				
900	275		500	
900		450	450	
930	350		500	
1000		500	500	6
Manualmente				
1050				**6**
1150		**500**	**600**	**8**

Velocità: Overclock manuale

Ci sono comunque anche voci di overclock a 1200 MHz e per chi volesse provare, sono disponibili sul mercato dissipatori a 7 euro, in grado di diminuire di qualche grado la temperatura del processore.

Oppure se avete ancora in cantina dei vecchi PC .. staccategli la ventolina, costerà ancora meno :)



Memoria: Condivisione memoria CPU/GPU

Cambio allocazione della memoria CPU/GPU

Di default Raspberry Pi parte con 64MB di ram per la GPU. Installando un sistema server possiamo diminuire questa memoria a 16MB

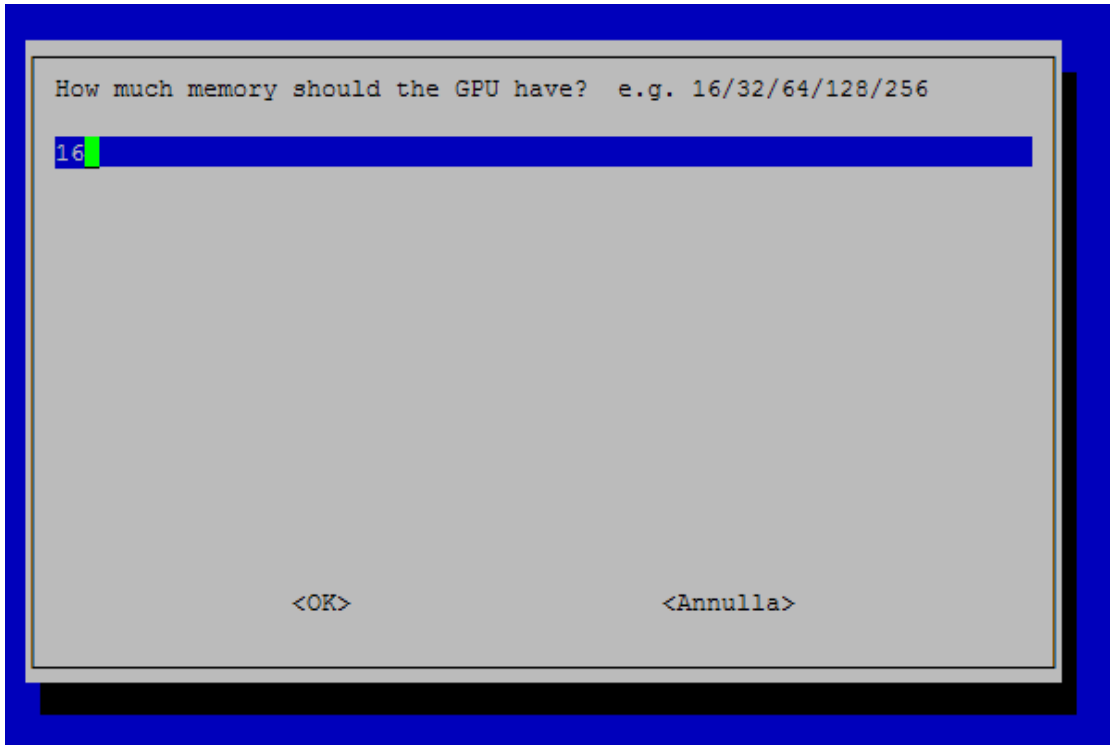
Possiamo modifiare

```
/boot/config.txt
```

```
gpu_mem=16
```

o usare

```
sudo raspi-config
```



Ottimizzazione servizi esistenti

```
sudo nano /etc/inittab
```

Al netto delle rimozioni fatte su Raspbian, possiamo ancora ottimizzare diminuendo i processi per le connessioni terminale, passando dalle 6 (default) a 2 .

```
1:2345:respawn:/sbin/getty --noclear 38400 tty1

2:23:respawn:/sbin/getty 38400 tty2

#3:23:respawn:/sbin/getty 38400 tty3

#4:23:respawn:/sbin/getty 38400 tty4

#5:23:respawn:/sbin/getty 38400 tty5

#6:23:respawn:/sbin/getty 38400 tty6
```

Consuntivo PRE/POST ottimizzazioni

Aumento velocità di elaborazione passando da 700 a 1000 MHz = 50%
Aumento di memoria = 60%

```
pi@piserver ~ $ free
             total       used       free     shared    buffers     cached
Mem:        497504      58816     438688          0      15588      26976
-/+ buffers/cache:      16252     481252
Swap:       102396          0     102396

pi@pinodo2 ~ $ free
             total       used       free     shared    buffers     cached
Mem:        448736     175880     272856          0      18368     137004
-/+ buffers/cache:      20508     428228
Swap:       102396          0     102396
```

Per chi vuole osare

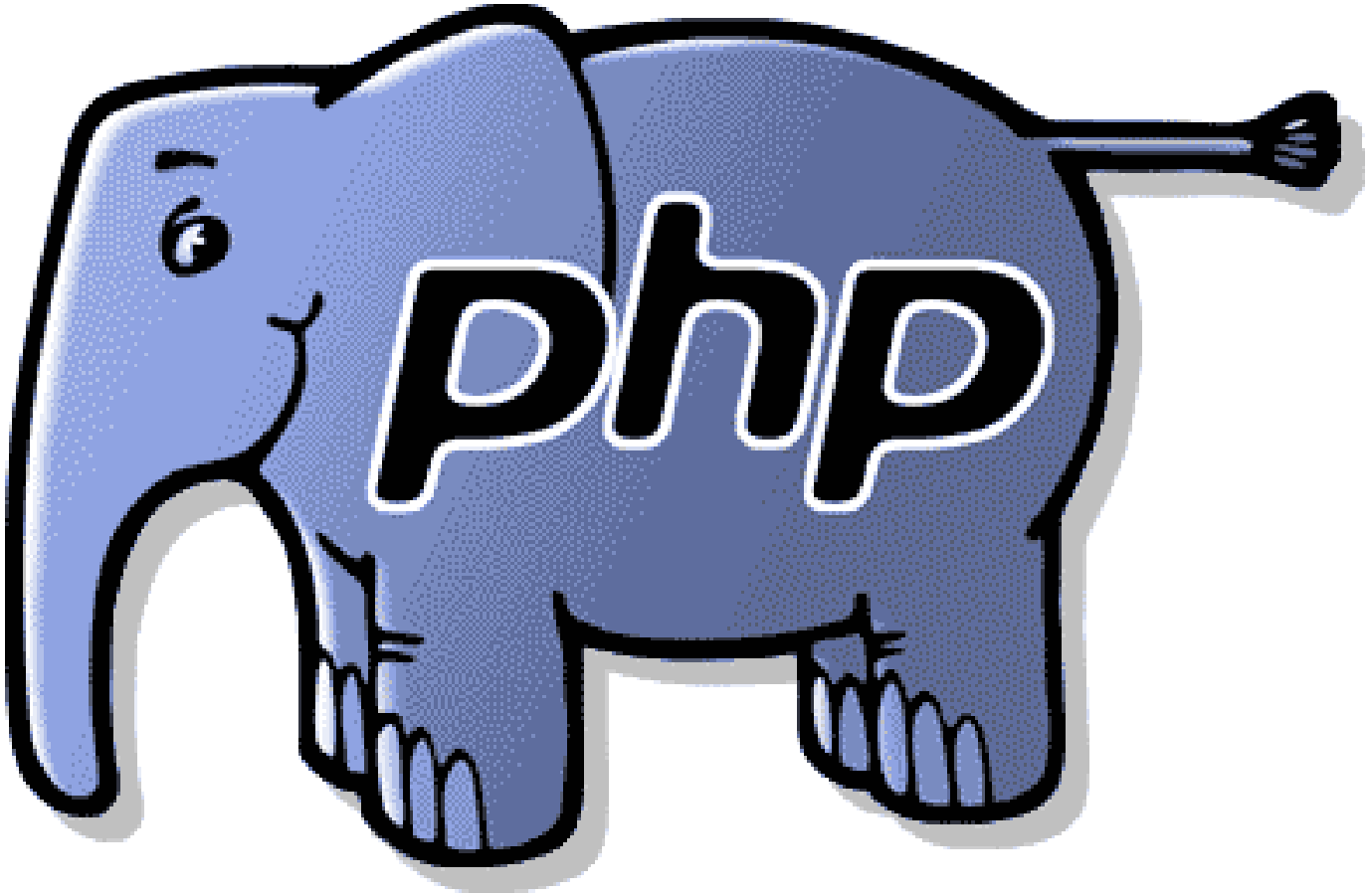
Possiamo **spremere** ancora di più Raspberry Pi

- staccando alcuni degli ultimi **servizi attivi** (p.e. rsyslog)
- passando ad 1 **sessione terminale**
- aumentando l'**overclock** e trovando un modo affidabile per raffreddare il processore
- acquistare delle SD più performanti (classe 10)

Ricordiamoci che l'**overclock** esagerato potrebbe rendere le SD utilizzate dal sistema operativo inutilizzabili.

Se questo non vi spaventa e avete già bruciato qualche processore (so che per qualcuno che legge queste slide non sarebbe la prima volta) allora potete osare.

Fino alla configurazione indicata, **funziona tutto**, oltre, basta provare :)



Installazione stack LAMP

Ora che abbiamo ottimizzato Raspberry Pi, possiamo passare allo stack LAMP

Di default, lo **stack LAMP non è installato** su Raspbian.

Essendo un sistema Linux, l'installazione dello stack LAMP è identica a quella che si farebbe su un qualsiasi server Debian non Raspberry Pi

Installiamo lo stack LAMP + PHPMyAdmin

```
sudo apt-get install apache2 \
                     php5 \
                     mysql-server \
                     libapache2-mod-php5 \
                     php5-mysql \
                     php5-fpm \
                     mysql-client \
                     phpmyadmin
```

Linux	Jessie kernel 3.10.36
Apache	2.4.9
MySQL	5.5.35
PHP	5.5.10
PhpMyAdmin	4.1.12

NGINX

```
sudo apt-get install nginx
```

MONKEY

```
sudo nano /etc/apt/sources.list
deb http://packages.monkey-project.com/primates_pi primates_pi main
apt-get update

sudo apt-get install monkey              \
                     monkey-liana        \
                     monkey-logger       \
                     monkey-dirlisting   \
                     monkey-cgi          \
                     monkey-fastcgi      \
                     monkey-mandril      \
                     monkey-cheetah      \
                     Monkey-auth
```

LIGHTTPD

```
sudo apt-get install lighttpd
```

Stress test e Bilanciatore

Per valutare le prestazioni del sistema sono necessari 2 tool

- Programma di Stress Test. Nel mio caso ho installato e compilato siege-3.0.5 su Cygwin (lo so sono un programmatore Windows)

```
wget http://www.joedog.org/pub/siege/siege-3.0.5.tar.gz
```

- Un bilanciatore di carico, in questo caso l'ho utilizzato direttamente sul nodo Master del cluster di test

```
wget http://haproxy.1wt.eu/download/1.5/src/devel/haproxy-1.5-dev19.tar.gz
make TARGET=linux2628 USE_ZLIB=1
```

Ho scelto di non utilizzare haproxy 1.4 presente in Raspbian Pi, dato che non è presente una importante feature : la **compressione GZIP**

Ambiente di test

Per i test ho optato per 2 file diversi, in modo da verificare le ottimizzazioni possibili in caso di **risorse testuali** (html, css, js, etc)

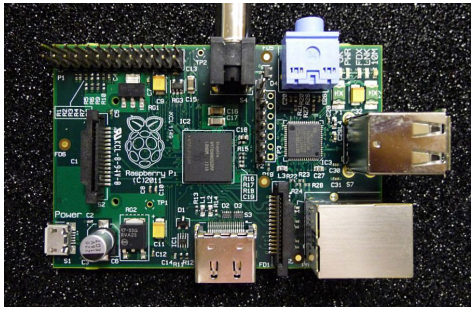
- <?php phpinfo() ?>
- Il sorgente html di phpinfo

Ho **volutamente evitato** di parlare di risorse non testuali già compresse (**immagini, filmati**) in quanto occorrerebbe una trattazione apposita, e la realizzazione di un server specializzato nel delivery di risorse statiche (la classica **installazione CDN**)

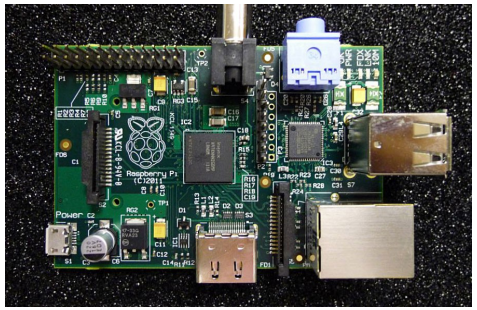
I test sono stati fatti tutti mettendo in concorrenza **5 processi sullo stesso server**, dato che stiamo pensando ad un ambiente web di test o dedicato a piccole aziende, dove non vengono superati i **5000 unique user al giorno**

Ambiente di test

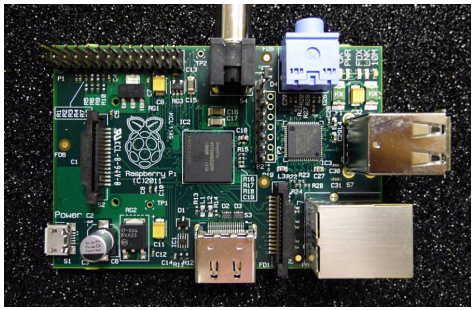
Nodo server – Raspberry Pi (made in China)
haproxy



Nodo 1 (made in UK)
LAMP modificato



Nodo 2 (made in UK)
LAMP con i default



Nodo2 a default: GZIP compression

780 pagine per **6.81**MB di traffico in 1 minuto

```
$ siege -q -b -t 1m -c5 http://192.168.2.22/phpinfo.php -m pinodo2 -lapache.log

Lifting the server siege...      done.

Transactions:                    780 hits
Availability:                 100.00 %
Elapsed time:                  59.93 secs
Data transferred:               6.81 MB
Response time:                  0.38 secs
Transaction rate:              13.01 trans/sec
Throughput:                     0.11 MB/sec
Concurrency:                    4.99
Successful transactions:         780
Failed transactions:               0
Longest transaction:            1.02
Shortest transaction:           0.19


top - 01:27:59 up  3:20,  1 user,  load average: 3,66, 1,84, 0,85
Tasks:  72 total,   7 running,  65 sleeping,   0 stopped,   0 zombie
%Cpu(s): 92,6 us,  5,2 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  2,3 si,  0,0 st
KiB Mem:    448736 total,   268056 used,   180680 free,    33304 buffers
KiB Swap:   102396 total,        0 used,   102396 free,   165432 cached
```

Nodo1: senza GZIP compression

1754 pagine per **88.44**MB di traffico in 1 minuto

```
$ siege -q -b -t 1m -c5 http://192.168.2.21/phpinfo.php -m pinodo1 -lapache.log

Lifting the server siege...      done.

Transactions:                   1754 hits
Availability:                 100.00 %
Elapsed time:                  59.27 secs
Data transferred:              88.44 MB
Response time:                  0.17 secs
Transaction rate:              29.59 trans/sec
Throughput:                     1.49 MB/sec
Concurrency:                    4.99
Successful transactions:        1754
Failed transactions:               0
Longest transaction:            0.43
Shortest transaction:           0.11

top - 01:29:50 up  3:22,  2 users,  load average: 2,47, 1,31, 1,15
Tasks:  73 total,   9 running,  64 sleeping,   0 stopped,   0 zombie
%Cpu(s): 73,7 us, 12,8 sy,  0,0 ni,  0,3 id,  0,0 wa,  0,0 hi, 13,2 si,  0,0 st
KiB Mem:    448736 total,   296424 used,   152312 free,    14848 buffers
KiB Swap:   102396 total,        0 used,   102396 free,   226832 cached
```

Apache Test

Usando Haproxy su un nodo master, ho provato a trasferire sul master la compressione del dato e scaricare il nodo dal consumo di CPU

Dati di partenza

```
        Date & Time,  Trans, Elap Time, Data Trans MB,  Resp Time
**** pinodo2 php gzip ****
2013-11-26 01:23:18,    765,     59.12,             6,       0.38

**** pinodo1 php ****
2013-11-26 01:21:30,   1809,     59.64,            91,       0.16
```

Overhead di haproxy

```
**** hapiserver pinodo1 php ****
2013-11-26 01:39:35,   1736,     59.94,            87,       0.17
```

Compressione lato server: prestazioni doppie rispetto al singolo nodo

```
**** hapiserver pinodo1 php gzip ****
2013-11-26 02:25:39,   1538,     59.44,            15,       0.19
```

Al posto che usare un output generato da PHP, proviamo ora ad usare direttamente una pagina HTML con lo stesso testo, per capire l'overhead generato da PHP

Pagina di partenza

```
        Date & Time,  Trans, Elap Time, Data Trans MB,  Resp Time
**** pinodo1 php ****
2013-11-27 02:54:02,   1757,      59.95,           88,        0.17
```

Pagina html

```
**** pinodo1 htm ****
2013-11-27 02:49:27,   5446,      59.52,          279,        0.05
```

Pagina php nodo1 compressa lato server

```
**** haserver pinodo1 php gzip ****
2013-11-27 02:46:21,   1479,      59.90,           15,        0.20
```

Pagina html nodo1 compressa lato server

```
**** haserver pinodo1 html gzip ****
2013-11-27 02:47:51,   2808,      59.86,           29,        0.11
```

Pagina php con 2 nodi, compressi lato server: 3 volte il singolo nodo

```
**** haserver php 2 nodi ****
2013-11-27 03:13:07,   2580,      59.47,           26,        0.12
```

Nginx 1.4.7

Nginx ha dato dei risultati diversi rispetto ad Apache. Per esempio, la perdita di prestazioni all'atto di una compressione GZIP è praticamente assente.

Vengono invece degradate, come ci si aspetterebbe, le prestazioni di HTML puro.

Rispetto a Apache, l'uso di PHP è attraverso l'interfaccia PHP5-FPM (FastCGI Process Manager)

```
        Date & Time,  Trans, Elap Time, Data Trans MB,  Resp Time
**** nginx pinodo2 php ****
2013-11-29 02:27:02,   1200,      59.52,           83,       0.25

**** nginx pinodo2 php gzip ****
2013-11-29 02:37:33,   1265,      59.53,           21,       0.23

**** nginx pinodo2 htm ****
2013-11-29 02:29:31,   4660,      59.95,          326,       0.06

**** nginx pinodo2 htm gzip ****
2013-11-29 02:38:46,   2914,      59.44,           53,       0.10
```

Monkey test

Monkey non dispone di una compressione GZIP, quindi i dati che restituisce sono sempre quelli presenti su disco. I risultati sono similari a Apache:

```
         Date & Time,  Trans, Elap Time, Data Trans MB,  Resp Time
**** monkey piserver php  ****
2013-11-29 03:35:13,   1502,      59.20,          104,       0.20

**** monkey piserver htm  ****
2013-11-29 03:36:41,   4456,      59.10,          332,       0.07
```

Una curiosità. Sul sito:

http://www.monkey-project.com/benchmarks/raspberry_pi_monkey_nginx

Vengono presentati dei benchmark che fanno pensare che Monkey sia più efficiente rispetto a Nginx.
IMHO il test non è veritiero, dato che dura 10 secondi e il file utilizzato per i test è un file **jpg**, dove la compressione è **chiaramente** inutile.
NB: file di questo tipo dovrebbe risiedere su una **CDN** e non su un server Monkey

```
siege -b -t10S -c200 http://localhost:PORT/linux.jpg
```

Lighttpd test

Lighttpd risponde in modo ancora diverso, non comprimendo i file PHP, ma solamente i file HTML statici.

In questo caso l'aumento prestazionale è molto importante, dovuto anche alla strategia di compressione di Lighttpd, che verifica lo stato di aggiornamento dei file statici e se risultano non aggiornati, restituisce una versione "pre" compressa presente all'interno di una percorso di cache.

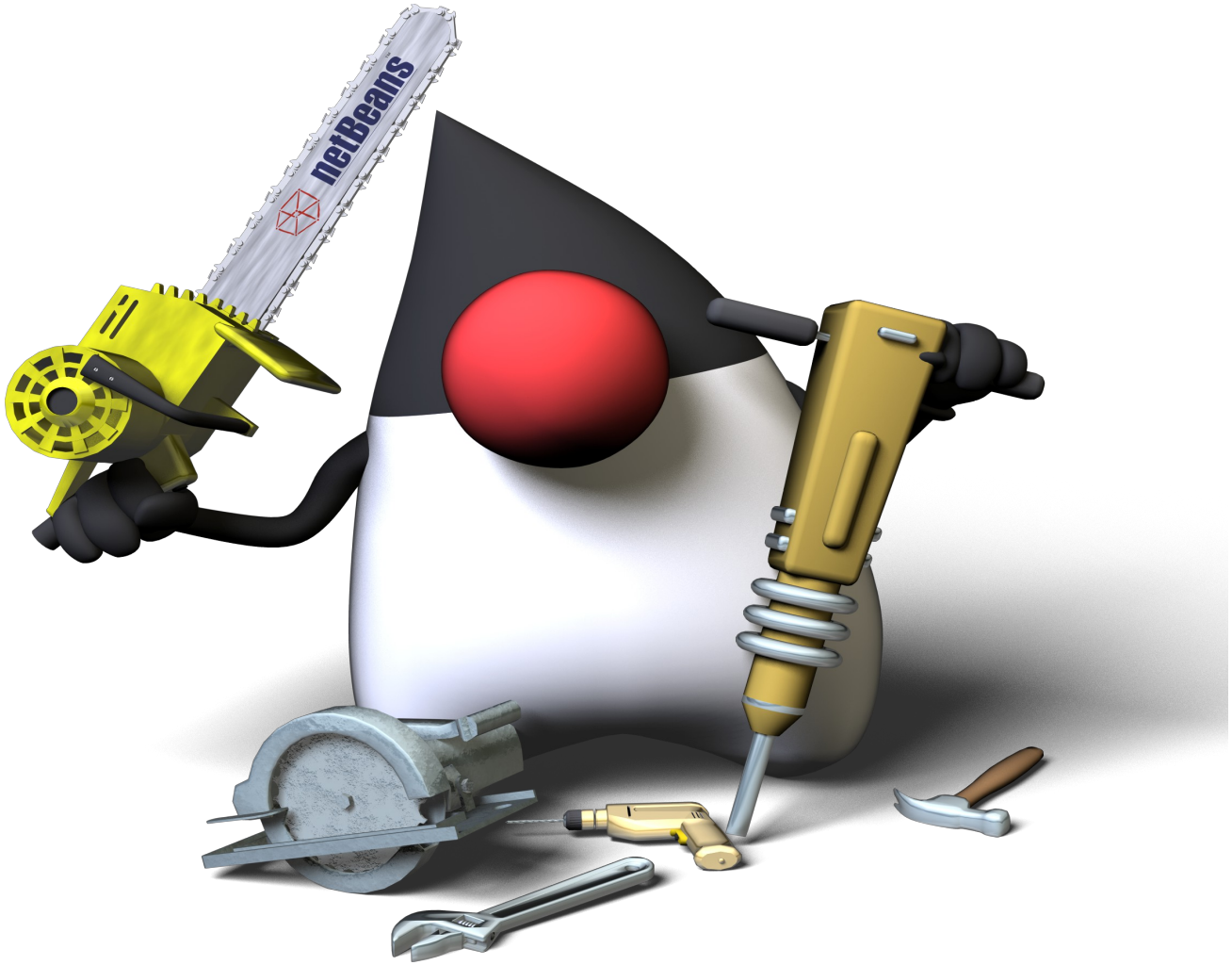
```
         Date & Time,  Trans, Elap Time, Data Trans MB,  Resp Time
**** lighttpd nodo2 php ****
2013-11-29 03:20:10,   1505,      59.13,           104,       0.20

**** lighttpd nodo2 htm ****
2013-11-29 03:21:29,  17575,      59.93,           157,       0.02
```

Conclusioni

Cosa abbiamo capito

- Valutare prestazionalmente PHP su Raspberry Pi può dare risultati molto diversi, in base al tipo di test che andremo a fare
- Alcuni server web sono privi di alcune funzionalità di base (**gzip**) che però possono essere integrate all'interno del bilanciatore
- Ottimizzando nel modo corretto, possiamo sfruttare parte della potenza del processore del bilanciatore, aumentando le prestazioni del nostro HW
- Per il tipo di server che andremo ad installare, non c'è una grande differenza di prestazioni da una soluzione all'altra


NetBeans

Java su Raspberry Pi

Prima del rilascio da parte di Oracle di un JDK specifico per Raspberry Pi, veniva utilizato OpenJDK

```
java version "1.6.0_27"
OpenJDK Runtime Environment (IcedTea6 1.12.6) (6b27-1.12.6-1~deb7u1+rpi1)
OpenJDK Zero VM (build 20.0-b12, mixed mode)
```

Da settembre 2013 possiamo installare un “vero” JDK

```
sudo apt-get install oracle-java7-jdk
```

```
java version "1.7.0_40"
Java(TM) SE Runtime Environment (build 1.7.0_40-b43)
Java HotSpot(TM) Client VM (build 24.0-b56, mixed mode)
```

Dal 18 Marzo 2014 è però possibile installare anche JDK 1.8

```
sudo apt-get install oracle-java8-jdk
```

java version "1.8.0"
Java(TM) SE Runtime Environment (build 1.8.0-b132)
Java HotSpot(TM) Client VM (build 25.0-b70, mixed mode)

Il primo test è stato quello di utilizzare un “evergreen” di Java: Tomcat versione 7 e 8.

Nonostante la lentezza, il sistema parte in un tempo che oscilla fra i 22 e i 26 secondi

Una volta installato, dopo la prima compilazione, il sistema risponde senza grossi problemi

Il secondo test è stato quello di utilizzare JBOSS

In questo caso è stato necessario modificare delle configurazioni per poterlo utilizzare dato che la memoria richiesta di default è troppo alta

Occorre quindi modificare i parametri

-Xms64m -Xmx512m

portandoli a valori più coerenti per Raspberry Pi

-Xms64m -Xmx128m

Il sistema parte in circa 150 secondi

Vediamo ora le prestazioni su uno stress test di 1 minuto

Con OpenJDK

```
         Date & Time,  Trans, Elap Time, Data Trans MB,  Resp Time
**** piserver jdk6 tc 7 ****
2013-11-29 04:13:40,     31,      59.03,             0,       8.82
```

Con JDK 7

```
**** piserver jdk8 tc 8 ****
2013-11-29 04:12:18,   2362,      59.09,            25,       0.12
```

Con JDK8

```
**** piserver jdk7 tc 8 ****
2013-11-29 04:17:11,   2507,      59.11,            26,       0.12
```

Conclusioni

Una volta partito, utilizzando un JDK 7 o 8, il sistema funziona.

Per poterlo utilizzare al meglio è però **necessario** procedere all'ottimizzazione della RAM utilizzata e possibilmente disaccoppiare il database, in modo da eseguirlo su un'altra macchina o un altro Raspberry Pi, dato che il quantitativo di RAM disponibile è veramente limitato

JAVA

Sicuramente Java non è il linguaggio migliore da utilizzare all'interno di un Raspberry Pi, ma grazie agli ultimi rilasci, ora è un linguaggio utilizzabile con prestazioni accettabili.
Fino a OpenJDK 6 le prestazioni erano decisamente scadenti

PHP

Risulta avvantaggiato dal limitato overhead che richiede al sistema per poter essere utilizzato, grazie a questo, non è impensabile utilizzare un server RaspberryPi con un intero stack LAMP installato e con tutte le applicazioni che ne conseguono

Tutto bello, ma se non avessi un Raspberry Pi?

Puoi emularlo con QEMU, utilizzando direttamente le immagini fornite dal **http://www.raspberrypi.org/downloads**

DEMO

Domande?

Come contattarmi

Matteo Baccan

matteo@baccan.it

http://www.baccan.it