Integrating Monitoring Data

...or how to get Management to approve your next gadget

By Christoph Sold

Since computers were invented, they were monitored: as a scarce resource computing power has always been a valuable thing. Even today, there is not nearly enough computing power available to solve all the numerical problems computers are easily applied to – let alone all those non-numerical problems you can imagine.

This paper deals with the requirements to further the development of monitoring and the effects this will have on software development and deployment. The effects of a concerted plan for monitoring will be shown, as well as how monitoring will help you to better predict your businesses needs.

Monitoring History

In the earliest days, arrays of *blinkenlights* were the tool of the day. Since then, many additional tools were invented. Some of them have survived the tides, others were swept into oblivion.

Although computer science has made a good deal of progress since then, the basic monitoring tools have not yet made as much progress. Many UNIX administrators still rely on basic tools like *ps*, *top*, or *netstat* as their primary tools of the trade. Some commercial UNIX systems have built upon those basic tools, e.g. Solaris *sar*, which allows for historical analysis, or AIX *Smitty*, which integrates configuration and monitoring. Commercial entities have developed various cross-platform monitoring tools such as *Borderware Patrol*, *Big Brother* (now from Quest Company), Lund *MetaView* or *Tivoli* (today a part of IBM). There are Open Source monitoring tools available, too, such as *Big Sister*, *Nagios*, or *NetSaint*.

Standards have been developed to help integrate monitoring data formats. The Simple Network Monitoring Protocol *(SNMP)* has been around for some time. The Application Response time Measurement *(ARM)* standard describes how monitoring data has to be logged and transmitted. Unfortunately, besides SNMP, there is no widespread standard to integrate various monitoring tools with each other. To make our life more miserable, even within SNMP there is no standard besides the numerical tree.

Why Monitor?

Like cars, defects can be fatal for your system. Thus, ***event (fault) monitoring*** needs to signal problems as timely as possible to minimize their effect. A well-known example of fault monitoring is the brake warning light in your car. If this thing lights up, you want to come slowly to a controlled halt to prevent dire consequences.

Performance monitoring continuously records system performance for deferred analyzation. This helps to predict system capacity needs. The fuel gauge in your car is a performace monitoring tool of sorts: it monitors your fuel efficiency.

Finally, ***service level monitoring*** watches for service level violations, giving a picture of your customers system perception to you. In modern cars, a built-in fault memory stores all faults. The car shop should analyze all the stored faults the next time you turn the car in.

Monitoring Tools Users

As in cars, the monitoring tools in the computing business have different customers.

Help desk workers as well as on-duty system administrators need ***event monitoring*** as the most important tool. Detecting problems before your customer calls is the killer application for event monitoring. A properly designed event monitoring system will help you to minimize the impact of your problem to the customer.

Performance monitoring helps your technical architects to predict your customers needs. Depending on system usage, it can be helpful for the system administration team to predict system utilization to some degree. Your sales people will also benefit, because knowing what your customer needs before he knows is always a good selling tool.

Service level monitoring is the tool of the trade for both your customer as well as for your sales and managerial people. Knowing the service level of your system enables your sales people to set the expectations of your customers into the right frame. In case of differences between you and your customers it is easy to check recorded service level data against your service level agreements.

Basic Monitoring Tools

Like any exact science, monitoring is based on repeatable, simple tools. Mathematics applied to this simple tools yields additional informa tion. To make your private crystal ball work, you have to pay extreme caution when implementing the basic building blocks.

Counting

Regardless of what you plan to measure, it basically comes down to counting. Be it the size of the average file in your hard disk, the packets down a pipe during a given period, or the seconds between two events, the basic process which has to be implemented is to count the basic unit of interest. This can be as simple as a variable incremented every time an event to measure occurs, or a simple routine counting the number of entities in a queue. There is one point you have to watch out for: Counts have to be *atomar*. If the number of entities changes while you count them, the count will be rendered invalid.

In addition, counts should be implemented with minimal impact. It is wise to invent a method to selectively disable counts not needed, such as a kernel variable, or to selectively enable only counts when needed such as in ipfw count rules.

Counts can be differentiated into *sum counts*, which integrate the number of units by adding them continuously until reset. Implementation is usually as simple as incrementing the counter each time the entity is detected.

Resetting the sum periodically at fixed intervals yields an *integrated count*, which helps to measure something over time. If you intend to record these, pay attention to record the count before the next reset occurs. Many scripts do exactly that to report on the traffic flow through *ipfw* packet filter. Mac OS X as well as Solaris provides *sar*, which records system usage statistics in five- minute- increments.

A third type of a count is the *event record*, which records the time an event occurred, eventually along with details what exactly happened then. *Syslogd* does this type of event recording. ipfw can be instructed to generate log events for syslogd, as well as Apache generates this type of log, which allows to calculate floating averages. On the down side more system resources are needed to store event records. ipfw implements limits to prevent against DOS attacks for this reason.

Queue length counts the number of items in a line of items to be processed. Since queues usually grow as your systems lag behind, queue length usually is a very good indicator for system overload. Well known queue length counts include network protocol I/O queues.

If only the sum of any entity is relevant, *total counts* are the tool of the choice. To help analyze entities which would overwhelm a total count, sampling into the data pool helps to provide insight. *Sampled counts* have to be implemented carefully: either have all of your samples synchronized, or the values will not sum up properly.

Throughput

Adding the time dimension to basic building blocks yields in throughput values. As mentioned above, *integrated counts* simply reset after a fixed amount of time. Count the number of events during a sliding period within an event record to get a *sliding integrated count*. Queue length over time yields *average queue length* counts. For queues, it is interesting to watch *queue length difference* over a fixed period of time.

Statistical Tools

Basic counting in place and properly unified, it is simple to apply standard statistical indicators such as variance, arithmetic or geometric mean values. Classical statistics are best adapted to jobs on historical data. In addition, they tend to be optimized for print media: Display a lot of static information in a small place with high resolution. They tend to print number forests, which makes interpretation on the fly difficult. Graphics such as box plots, scatter graphs or quantile-quantile-plots help to visualize these approaches.

NIST has developed a new approach to analyze statistical data, along with a tool to plot the lot.[1] The fresh approach they used to apply to engineering statistics is rather helpful when slicing and dicing through monitoring data woods.

---

[1] *NIST/SEMATECH e-Handbook of Statistical Methods*, `http://www.itl.nist.gov/div898/handbook/`, 2004.
The tool is named *dataplot*, available at `http://www.itl.nist.gov/div898/software/dataplot/homepage.htm`.

Basic Analysis Tools

Most monitoring tools are suited to provide real-time data to system administrators. They allow to glimpse into various aspects of your systems in real time. Others allow to analyze historically recorded data of one specific aspect of your systems.

The simplest tool is recorded basic monitoring data. To be of any help, all monitoring data should be recorded and archived. Mac OS X Server as well as Solaris monitor some essential performance data sets unconditionally during normal operation. For a short period of time (usually a few days), performance is sampled every five minutes and stored into the *sar* database. The database files are rotated daily, after a few days, they get overwritten. The *sar* utility allows to single out specific information as long as the database files are still intact.

Almost all descendants of *UNIX* still provide some file system quota mechanism along with the tools needed to monitor disk usage. While not recording anything when you‘re not over quota, they'll get verbose when you cross soft or hard quota limits. It‘s up to you to check disk quota logs against your policy.[2]

Network interfaces usually get recorded, too. netstat -i reports recorded network interface statistics. It‘s up to the kernel to record the packet counts. The interface netstat offers into your TCP/IP stack statistics resets the statistics every time you view them. Once viewed, the statistics are gone.

The tools mentioned above have some things in common: a back end records the data, while a front end lets you get at that data in human-readable form. Using back ends to record the data helps to minimize monitoring impact on your system. Calculate front end values only when they are needed.

A front end application like netstat, iostat or vmstat lets you access the recorded data when you wish to get at it. Back when network access was trusted in the days of the r-tools, secondary front ends to machine load were commonplace: xdm lets you display the load of remote servers will-

---

[2] Quotas can be a real PITA because often applications try to allocate files unknown to average Joe User. Ever tried to open a nicely compressed JPEG photo fresh from your digital camera with *GIMP*?

ing to let your X server connect. ruptime shows load data like the data displayed by w, ps, or top. All of those tools tap into the in-kernel load variables.

Unfortunately, even simple monitors like top differ in how they record monitoring information: on a machine with four CPUs, does a load of 1.14 mean there is one CPU working 100 %, another one is nearly idle, while two others are doing nothing at all, or are there all CPUs happily crunching numbers, while a little excess load waits for any CPU to become free? Both points of view are valid, and, unfortunately, both have been implemented in different operating systems. Remember to make sure things named the same are measured using the same algorithm before comparing the measured values.

Historically, minimal data was recorded to keep the impact on system performance as well as disk cost as low as possible. Today, there is no such thing than a history of machine loads. Recording Veritas Volume Manager managed file system performance for all your SAN volumes would be nice to analyze performance historically. Guess if it is available out of the box.

Monitor Displays

There is no such thing as a single monitor display fits all. Depending on the situation, you'll need different displays for different users in different situations. Having a ssh-to-SMS-Gateway may be a nice feature, but have you ever tried to recognize systat -vm output on a 40x4 mobile display?

In addition, different users of monitoring data want to view the data in completely different detail. Upper management is usually only interested if everything works within predefined parameters, while your system administrators need a lot more information to get the job done. Customers tend to ignore all monitoring data but end to end cycle times. Project managers are interested often only in the most damaging limit, which may change from minute to minute.

Defining Display Consumers

Before any monitor is designed, the users of the display have to be stated clearly. Each display has one primary consume as well as any number of secondary consumer groups. *Primary consumers* will need to examine the data in greater detail and at a higher frequency than *secondary consumers*.

Interactive consumers should be able to change the way data is displayed, while *passive consumers* need only to access a default data display. Sometimes this will include raw data exports to satisfy the need for interactive customers.

Defining the consumer groups helps to make clear who can ask for features in a new display, and - often more important who will have to live with what primary consumers defined. Make sure there is only one primary consumer to avoid conflicting interests.

Analyze Monitoring Baseline

After defining all your monitor user groups, define the baseline for all monitors. To do so, all monitoring data needs to be classified. For each monitor, the primary user requests resolution and retain period of monitored data. Thus, it will define how much storage this monitor needs while running. In most situations, an experienced systems administrator has to assist the customer during the definition phase.

The configuration of both the monitor as well as monitored object has to be recorded, too. Analyzing CPU loads while assuming the wrong hardware can produce results way out of the ballpark.

Data Compression Algorithms

Depending on your monitors needs, it may be possible to implement techniques to reduce space requirements of historical data.

Archiving trades access time for space. Database files, while fast, use notorious amounts of space to gain best access speed. Exporting and archiving older data saves space. A frog view reduces time resolution for older data. One simple algorithm to do this is to calculate mean values over a few measurement cycles and store them instead of the original data. Resolution Reduction transforms high resolution measurement values into low resolution, e.g. from float to single byte integer values.

Even if space is no concern, it may be wise to export historical data to keep access speeds of your live monitoring tools up to the job.

There is no way to reconstruct monitoring data once it has been deleted. Working with interpolated data will give wrong results, so be careful what you throw away.

Monitoring Data Interfaces

There is no such thing than universal software. Software, as well as hardware, will be replaced sooner or later, not teccessarily with something know to you today. Defining and documenting a data format as well as the interface between acquisition, back end, and front end will help you to retain valuable historical information when any piece in your monitoring systems chain changes.

Agree on Warning and Alert Levels

Monitors needing neither warnings nor alerts are rare things. Before starting to implement, collect all alert and warning level definitions from the customers. There may be monitors doing nothing else than warnings or alerts. Both system level and application level alerts need to be defined consistently .

Define an Priorization and Alarm Plan

Clearly define which level of alert needs what level of attention. Your alert plan should clearly state who has to be informed by whom. In case of disaster, there is no time to discuss who can help, too. Ideally, each monitor will be linked to its alarm plan.

For less dire situations, it is helpful for newcomers to have events already prioritized. Track all events, regardless how minor they seem. Having a record of lots of small problems with any device helps to convince management to replace it with something with lower maintainance cost.

Choose Presentation Formats

Depending on your data as well as your customers needs, choose presentation formats. Depending on your customers needs, this means there will be multiple monitors displaying the same data in different views in differing detail. Make sure each primary customer gets exactly the view she asked for, no matter how silly this may seem to you.

Implement the Data Acquisition Layer

After collecting all neccessary information shown above, coding can be started. Depending which tool you have choosen, this may be as simple as hacking your definitions into Tivoli through its console, or as complicated as patching away against packet filter code to collect raw data.

At this time, choose the monitoring method which gives you just the information you have been asked for at the cheapest cost.[3] Store the monitored data into the data store of your choice, and forget about it.

Remember to monitor the monitoring functions, too. Many a monitor has been grounded for too much false alarms, or for not reporting dire problems in time. Monitoring irregular cycles is especially difficult. Watch out for unusually long capture periods. If at all popssible, use an artificial heartbeat to make sure your monitor still works.

Integrate Information

There is no such thing than a luser-free SAN or an applicationless database. All of those things have their real-world use. Unfortunately, employees within departments tend to cling together, and easily view other departments as enemies. This often leads to internalization of information. Having a DBA Team in a full strength search for a performance leak caused by the Unix team moving the index table space from fast local hard disks to bigger, but slower NAS filers will lead to both: performance problems for your customer, as well as hostilities between teams.

Team people within various departments using the same hardware or software to prevent this kind of problems. Make sure both administrators as well as users of each subsystem know each other, as well as they should now about each others needs. To solve rising problems as soon as possible, shorten the line of communication between complimentary teams. To make sure both team know what the other speaks about both have to be able to look at each others monitoring displays.

Doing the Whiz: Predicting Future

Aside from the tools mentioned above there are a lot of tools available to predict the future of your systems: magic eight-balls, crystal balls, gut feeling, and throwing dices, to name a few. Comparing the cost of the former to the cost of well-implemented monitoring as described above makes those cheap gizmos even more attractive.

Applying modern statistical analyzation techniques to properly recorded monitoring data will lead to a prediction hit rate much higher than the average 50 percent by throwing coins. Choosing appropriate data collec-

---

[3] Depending on your needs, cost can be either minimal implementation cost or minimal impact on the monitored resource. Your call.

tion mechanisms will allow to apply the monitoring techniques in real-time.

This will put you into a much better picture to decide how your systems should be changed. Identified resources in excess of someones need may be better allocated to another use. Bottlenecks discovered may be solved before their impact punches through to the customer. Even better, analyzation of recorded incidents will show parts of your system with lots of small problems before they become big problems – and you'll have the paper to back your requests for new tools to widen the bottleneck.

Literature

NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/, 2004.

Toutenburg, Helge: Deskripvtive Statistik, Springer Berlin Heidelberg 2004