Making Everything Easier!™

BigPanda Special Edition

Alert Correlation FOR DUMMIES®

A Wiley Brand

Learn to:

- Combat IT alert overload
- Successfully automate IT operations
- Select an alert correlation solution that works

bigpanda

Alert Correlation FOR DUMMIES® A Wiley Brand

BigPanda Special Edition

by Allan Konar

FOR DUMMIES® A Wiley Brand

Alert Correlation For Dummies®, BigPanda Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN 978-1-119-27849-8 (pbk); ISBN 978-1-119-27850-4 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

We're proud of this book and of the people who worked on it. For details on how to create a custom *For Dummies* book for your business or organization, contact info@dummies.biz or visit www.wiley.com/go/custompub. For details on licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com.

Some of the people who helped bring this book to market include the following:

Project Editor: Susan Pink

Acquisitions Editor: Amy Fandrei

Editorial Manager: Rev Mengle

Sales Development Manager: Karen L. Hattan

Production Editor: Antony Sami

Table of Contents

Introduction

The last decade or so has seen significant changes driven by the integration of technology into every facet of our lives. Where once we might have looked in the want ads of our local paper, we now go directly to eBay.

Calling a cab?
Nope, we're going to Uber it.

Looking at *AutoTrader* magazine for a used car?
Autotrader.com.

Office furniture store?
Wayfair.com.

Watch a sitcom on TV?
Hulu.com.

Over the last several years, we have moved towards the use of an expanding range of platforms for these services. We're no longer bound to a PC when we consume these services. Mobile phones, tablets, set-top boxes, gaming consoles, and even Internet-connected appliances provide us interfaces to goods and services in ways we never could have imagined.

This shift towards a software based, on-demand economy has necessitated (and been enabled by) a vertical scaling of the systems and services that run our businesses.

The availability of public, private, and hybrid cloud-based services has been a key driver in scaling and accelerating our business services. However, this has led to a dramatic fragmentation of systems that are all expected to interoperate seamlessly. Where we once might have run a simple PHP-based shopping cart on our website using a single server with a standard LAMP (Linux, Apache, MySQL, PHP) stack, we have now moved steadily towards decomposition of service elements into microservices.

Fast
Good
Cheap

Not only are things getting more complex and fragmented, they are also moving at a faster pace than ever before due to agile development methodologies. Application services are continuously updated across heterogeneous platforms, deployed at scale. And our customers expect consistent experiences whether they are using our website, an embedded web app, a living room device, or mobile phones and tablets.

Just let that sink in.

We are simultaneously seeing

- The fragmentation of platforms, including public/private cloud-based, virtual, and bare-metal platforms
- The need to support unprecedented scale in terms of concurrency and the size of applications
- An accelerating speed of development driven by agile methodologies

That's a lot of moving parts, complexity, and potential fragility, and downtime has quantifiable costs. We're not just talking opportunity costs here — real money is at stake.

Visibility is key to managing your network, storage, and computing assets to ensure continuously available services, right?

Sure — unless, like most IT teams, yours is drowning in data.

When you're under a constant deluge of alerts originating from heterogeneous systems and best-of-breed management solutions, you run the risk of IT alert fatigue. When your eyes start to glaze over while trying to identify the potential issues indicated by a large number of alerts, you run the risk of missing something critical to your business.

So how do you separate the wheat from the chaff? You correlate alerts.

By identifying related alerts from multiple sources, determining their relationships, and presenting them as a single event, alert correlation turns mounds of data into concise, actionable intelligence. Easy enough, right?

Wrong. This type of event correlation is much harder than you might think. The data that you rely on comes from many sources in disparate formats, often at a high volume. Add a constantly shifting infrastructure landscape — including never-ending cycles of updates and upgrades, vendor changes, and systemic moves to cloud-based services — and the process of consistently relating events into actionable intelligence becomes a fast-moving target.

About This Book

If you lead a team, are part of a team, or are singularly responsible for the day-to-day operations of enterprise critical systems, this book is for you. In *Alert Correlation For Dummies,* you'll find an overview of the challenges associated with maintaining optimal operations of these systems.

Ultimately, the goal of this book is to help you improve your teams' effectiveness by better understanding the real-world solutions to the challenges associated with operations and alert management. You'll explore using alert correlation to help focus your teams and enable them to operate more efficiently.

Icons Used in This Book

This book uses the following iconography to call your attention to items that are useful, might help you fall asleep at night, or are the functional equivalent to the classic, "For your safety while on the ride, please keep your hands, arms, and legs inside the vehicle at all times."

The tip icon indicates a piece of advice that has been garnered from practical experience. Consider it a word of wisdom from the Tao of Alert Correlation.

This icon signifies something that you should specifically note. If you were studying for a test, you'd break out your highlighter for this text.

This icon highlights something technical that not everyone will want to explore. If you like to know how things tick, or need help falling asleep, this text is for you.

Warning, Will Robinson! Learn from the mistakes of others.

Chapter 1

Alerting Overload (aka "The Problem")

In This Chapter

- ▶ Gnōthi seauton ["know thyself"]
- ▶ Alerts, alerts, everywhere
- ▶ One day you run everything, and the next day you run like a dog
- ▶ Buy the ticket, take the ride

Changes in customer usage patterns and expectations have driven an acceleration in application development cycles and fragmentation in system and service architectures. This has led to an unprecedented scale of IT alerts and events that operations teams must contend with on a daily basis.

In this chapter, you explore the drivers, effects, and costs of these trends.

Seeing Is Believing

Public/private clouds, containers, and microservices are proving to be significantly more flexible business-critical service building blocks than traditional monolithic solution infrastructures. The need for operational visibility grows in direct relation to the adoption of these heterogeneous critical system components.

As such, you'll find more tools to manage critical infrastructure, services, and applications than you can shake a stick at. With network and system monitoring, application and web performance monitoring, and log management tools, there is no lack of options for tools to provide visibility into the health of your production environments. This is a direct reflection of the seismic shift in how applications and services themselves are built.

TIP

In pol*IT*e society, we describe the availability of a plethora of monitoring applications as "Today's monitoring stack is rich." This abundance makes it difficult to reliably know your options. Check out MonitoringScape (`https://bigpanda.io/monitoringscape/`), a community-driven resource to help you stay on top of all of today's available monitoring tools.

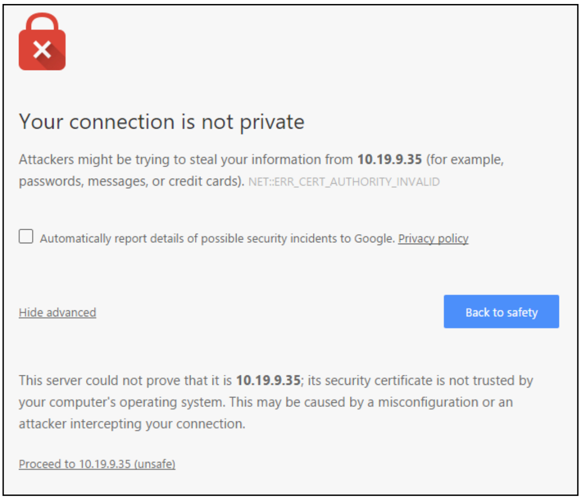
Monitoring tools are a critical part of the production lifecycle. They help you prevent issues before they affect customers by detecting and resolving faults faster. They also alert you when system components don't behave as intended, such as when the available storage on a server is too low, application latency is too high, the network connectivity is bad, or database files are corrupt.

What's the Catch?

In recent years, the number of alerts that companies experience in service-critical production environments has grown by orders of magnitude. IT and DevOps teams have been limited to utilizing traditional incident management approaches that have not evolved with the changing environment. Operations teams are forced to handle hundreds, and even thousands, of alerts every day.

Think about how often users ignore browser warnings about invalid SSL certificates, such as the one in Figure 1-1. A study of user decisions after seeing browser security warnings presented at the 22nd USENIX Security Symposium in 2013 found that people clicked through SSL warnings 33.0% to 70.2% of the time.

This failure to respond to security warnings is a function of *habituation,* a form of learning in which you unconsciously become desensitized to a stimulus after repeated exposure.



Your connection is not private

Attackers might be trying to steal your information from **10.19.9.35** (for example, passwords, messages, or credit cards). NET::ERR_CERT_AUTHORITY_INVALID

☐ Automatically report details of possible security incidents to Google. Privacy policy

Hide advanced

Back to safety

This server could not prove that it is **10.19.9.35**; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

Proceed to 10.19.9.35 (unsafe)

Figure 1-1: An invalid certificate is a commonly ignored warning.

The ECRI Institute's "Top 10 Health Technology Hazards" reports from 2010–2015 identify alarm hazards as the number one danger to patients from the medical community.

While alert-overload-induced alarm fatigue doesn't typically present life-or-death dangers as it does in the medical profession, to IT operations groups, it may introduce service downtime and reduced productivity, costing companies millions.

This Is Not Your Mother's Application Environment

The roots of the current situation stem from three major trends in application development and delivery:

- The popularization of agile development methodologies, such as Scrum and use of automation tools
- The adoption of service-oriented architectures
- Virtualization, public/private clouds, and containers

The quick and the dead

Consumer expectations of continual feature delivery and the need to remain innovative in an increasingly competitive global market have led to the widespread adoption of agile development methodologies, such as Scrum, and automation solutions, such as Jenkins and Chef, to speed up the software development lifecycle. In short, engineering groups are being pushed to move faster to deliver new capabilities by companies that are concerned about their products becoming outdated or obsolete. CI (continuous integration) and CD (continuous delivery) tools address this pressure, enabling the testing and deployment of code to production in minutes, rather than months, resulting in high volumes of changeover in a short time.

Service with a smile

Microservices, the modern evolution of SOA (service-oriented architectures), have become a popular method for scaling services both horizontally and vertically. By defining discrete application components that provide services to other distributed autonomous components, incremental changes may be introduced continuously, enabling the addition of features and functions, dovetailing comfortably with DevOps. The integration of heterogeneous platforms is addressed via technology-agnostic network communications and protocols. The challenge is that the decomposition of services may result in highly fragmented applications. When an application fails, how do we isolate which component, or connection, is the root cause?

Virtually any time, anywhere

The introduction of virtualization, public/private clouds, and containers have blurred the lines between applications, services, and platforms even further. Applications that might once have been hosted on a fleet of servers behind load balancers and connected via routers and switches may now be logically bundled and deployed on commodity hardware in data centers. Resources such as CPU, memory, storage, and network interfaces are shared across all hosted services in virtualized and cloud-based environments. Management and orchestration of these resources is critical. Any given function may starve cohosted virtual appliances.

With application functions no longer confined to discrete on-premise hardware appliances, service failures become much noisier and more complex to troubleshoot.

While these trends have enabled rapid development and delivery of highly scalable responsive applications across myriad platforms, they have also introduced significant complexity.

The math is simple:

> more components × more platforms × more changes = **monitoring data explosion**

The volume of alerts expands by several orders of magnitude in this scenario. Even SMBs (small and medium-sized businesses) may experience tens of alerts a day, with larger enterprises seeing thousands.

That’s a lot of chaff to separate from the wheat.

Taking Care of Business

Environments are getting more complex. Alerts are coming in fast and furious.

So what? Isn’t that what your teams get paid for? What’s the big deal?

The big deal is that alert overload has a concrete, measurable effect on the business. Several classes of costs are associated with alert overload, including the following:

- Bloated, underutilized IT operations teams
- IT operations staff retention and replacement
- Loss of revenue
- Customer churn (turnover)

He ain't heavy, he's my brother

The typical response when IT ops is drowning is to throw bodies at the problem. Organizations staffing for peak times often find team members sitting on their hands between emergencies, a clearly inefficient use of resources.

The traditional incident response loop (detection, triage, investigation, and remediation), as shown in Figure 1-2, proved effective when organizations dealt with a handful of incidents requiring no more than minutes to days to resolve.

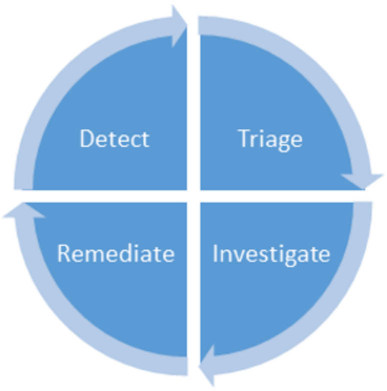


Figure 1-2: The traditional incident response loop.

When faced with alert overload, however, the cycle breaks. There simply aren't enough hours in the day to investigate and remediate (or escalate) issues when faced with hundreds or thousands of them. This leads to a continual expansion of operations teams (NOCs, IT admins, Level 1/2 engineers) until they outnumber the staff associated with your core business.

An interesting example of this kind of infrastructure investment is AWS (Amazon Web Services). Amazon.com began standardizing and automating their retail computing infrastructure in 2003. By 2006, they officially launched AWS, selling virtual servers as a service. What began as an attempt to simplify their infrastructure escalated to the point that they were able to add an entirely new line of business. Most companies don't have the available resources to build tools orthogonal to their core business, and instead rely on third-party software to build their customer-facing solutions, in addition to their service management infrastructure.

In their "Top Ten Pain Points of Operating Networks" report, Aviat Networks estimates that network operation staffing costs $120 billion per year globally. These OPEX (operational expenditures) are just the tip of the iceberg.

Think of the children

It was once common for employees to spend large swaths of their careers at a single company. But overwhelming day-to-day work conditions contributed to a bellwether change in the average length of tenure across industries, affecting IT operations particularly heavily.

In IT ops, occupational burnout, as recognized in ICD-10 (International Statistical Classification of Diseases and Related Health Problems, revision 10), is not uncommon. Exhaustion, lack of motivation, frustration, and cynicism are often worn as badges of honor. IT staff members seem to have coffee running through their veins.

This level of stress, however, is not sustainable. It's difficult to face the day expecting hundreds of alerts and thousands of lines of logs. Worse yet is being woken up in the middle of the night by a text message to respond to an event, with the expectation of being able to make a quick context switch and come to full alertness instantly.

According to PayScale's "Companies with the Most and Least Loyal Employees" report for 2014, the median employee tenure is 3.7 years for more than half of all Fortune 500 companies but only 1.1 years for IT groups.

Between recruiting, interviewing, and training, the cost to replace a knowledge worker can be significant.

In the IRLE working paper, "Employee Replacement Costs," the cost of replacing a worker is reported to be "as high as $7,000 for professional and managerial employees."

Kevin Kruse, in "What Is Employee Engagement," points out that there is another opportunity cost associated with employee engagement:

> *Engaged Employees lead to . . .*
>
> > *higher service, quality, and productivity, which leads to . . .*
> >
> > *higher customer satisfaction, which leads to . . .*
> >
> > *increased sales (repeat business and referrals), which leads to . . .*
> >
> > *higher levels of profit, which leads to . . .*
> >
> > *higher shareholder returns (i.e., stock price)*
>
> *As former Campbell's Soup CEO Doug Conant once said, "To win in the marketplace, you must first win in the workplace." Employee engagement is the key to activating a high performing workforce.*

Real-world costs are associated with losing valuable employees.

Oops! I did it again

Companies will not always have sufficient resources available to respond to every issue. At best, this means they have to constantly prioritize. For example, are you more worried about database response times or your web-based storefront's UX (user experience)?

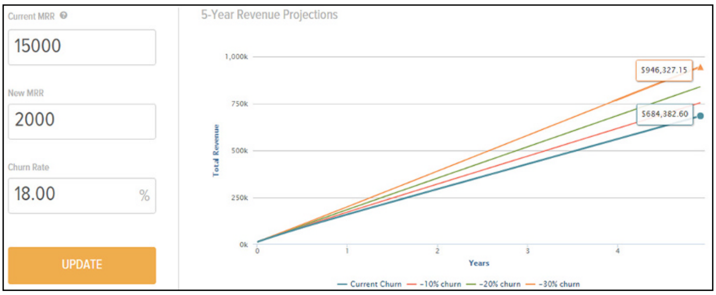
The worst-case scenario is one in which operations staff do not notice performance degradation — due to the high signal-to-noise ratio — until a service outage occurs. But what is the cost of downtime to your business reputation and customer loyalty?

WARNING! In September 2010, Virgin Blue suffered an outage to their Navitaire check-in system (hosted by Accenture), leading to 11 days of service interruptions and downtime and 130 cancelled flights and delays for more than 60,000 passengers. In a statement to the Australian Stock Exchange, Virgin Blue indicated that, "An initial assessment of this interruption shows an estimated pre-tax profit impact of $15–20 million." This failure didn't just affect Virgin Blue's reputation and bottom line revenue; it also cost Navitaire in the form of a lawsuit settlement for an undisclosed amount.

When alerts are not dealt with in a timely manner, the result may be unplanned service outages, which can cost millions, as in the preceding example. Add the loss of productivity, when critical systems are not available, and the costs continue to skyrocket.

Alan Arnold ("Assessing the Financial Impact of Downtime") writes that, on average, businesses lose between $84,000 and $108,000 (US) for every hour of IT system downtime, according to estimates from studies and surveys performed by IT industry analyst firms. The average total cost of unplanned application downtime per year for the Fortune 1000 is between $1.25 and $2.5 billion. The average hourly cost of an infrastructure failure is $100,000 per hour. The average cost of a critical application failure per hour is $500,000 to $1 million, according to Stephen Elliot in "DevOps and the Cost of Downtime: Fortune 1000 Best Practice Metrics Quantified."

Beyond a significant loss of revenue, the cost of the brand damage due to negative customer experiences, in the form of churn, may be significant. Consider the example in Figure 1-3, created using the churn impact calculator from churn-rate.com.


Current MRR 15000

New MRR 2000

Churn Rate 18.00 %

Figure 1-3: The cost of customer churn.

By reducing customer churn by 30%, the net revenue recovered over a five-year period was just shy of $262,000 (approximately 38%). You can find a range of estimates of the cost to reacquire a lost customer, but most are six to seven times the cost of retaining existing customers.

Add this cost to the opportunity cost in terms of lost revenue, and the numbers grow fairly quickly.

Chapter 2

Operational Landscape

In This Chapter

- Don't touch the sides
- And in this corner. . .
- Have you tried turning it off and on again?
- Ah, Houston, we've had a problem
- NOC, NOC. Who's there?
- Can't see the forest for the trees

It may be hard to believe, but there was a time when developers and support staff were two distinct groups that rarely interacted. More often than not, functionally separate groups were individually responsible for development, QA, support, and IT operations. Traditional operations functions centered on network management with centralized staff and tools.

The services and structures required to support customer interactions have changed radically over the past few decades. This means a significant shift in development and operations. Business applications have become critical infrastructure and are treated as such.

The line between network and application management has disappeared and led to greater teamwork between operations and development staff. This partnership is crucial to success given the exponential growth of resources to be managed and the density of messaging that infrastructure originates. Concurrent with this shift in relationships is the need to move away from traditional centralized monitoring and to employ greater automation to separate the wheat from the chaff of alerts.

Don't Touch the Sides

IT (information technology) operations are the people, processes, and services responsible for the ITSM (IT service management) associated with the ongoing control and maintenance of an organization's applications and supporting infrastructure.

Typically, IT operations success is measured by service quality and availability as outlined by an SLA (service-level agreement) and constrained by budget. In the simplest terms, IT operations monitor and control all operations related to IT infrastructure services. Traditionally, these operations groups worked in relative isolation from development teams outside of ticket escalations requiring bug fixes. This model was inherited from the telecom days of network infrastructure management.

And in This Corner

It's impossible to engage in a conversation about ITSM without hearing the buzzwords *ITIL* (IT Infrastructure Library) and *DevOps* (development operations). Both represent a framework or set of best practices with the goal of improving how we run IT operations. More often than not, the shared perception in the industry is that these two prevailing methodologies are at odds with one another.

ITIL focuses on *aligning IT with business goals*. DevOps focuses on *greater agility and reliability*. These goals may easily complement one other.

ITIL advocates clear roles and structured business processes. DevOps advocates higher reliance on automation and communication between IT and development organizations. Both target closer collaboration across the enterprise.

Of course, there is no one perfect "one size fits all" blueprint for how to structure and operate the development and operational functions of your business. Organizations must choose the aspects of each discipline that they want to adopt.

Try Turning It Off and On Again

IM (incident management) is an ITSM component targeting the timely restoration of normal operation to minimize the effect of service faults on business operations. The goal is to maintain the best possible levels of service availability and quality.

The ITIL framework identifies the process of responding to an incident:

- **Detection:** The discovery of a service outage or a defect that might lead to an outage
- **Classification:** The characterization of the urgency of incidents and who should own resolution.
- **Investigation:** The identification of the cause of the outage. What's the fastest way to restore service?
- **Resolution:** The application of a fix to restore service. Note that this may involve a stopgap measure, not a permanent fix. The goal is to bring back service first and foremost.
- **Recording:** Documenting the resolution, for post-mortem investigation and to accelerate resolution if the fault recurs. How can I prevent this fault from recurring, or reduce the time to service resolution if it does recur?

TIP

DevOps puts a great deal of focus on automation to remove challenges to offer services at scale. What might be a minor annoyance for a lightly utilized application could prove to be a significant time sink after heavy adoption. In the same vein, it makes sense to focus on automating the incident response process as just described.

Here are a few pointers to help you down the path to automation:

- **Detection:** Use monitoring tools, not your staff, to detect problems across your stack (system monitoring, application monitoring, log management, and user monitoring).
- **Classification:** Most issue types can be automatically classified, at least for the purpose of routing and prioritization. Ensure that issues are automatically routed to the appropriate escalation team based on the service and problem type. Additionally, don't rely on monitoring severity to prioritize. (For example, monitoring severity will tell you your storage level but not whether this storage unit is part of a critical service.) Identify your most critical services in terms of business impact, and make sure you can isolate high-severity issues when they occur.
- **Investigation:** Make investigation information as readily available as possible. Think "push" instead of "pull." When critical investigation information is pushed to the level-1 troubleshooter instead of that person having to actively look for it, TTR (time to resolution) is cut by an order of magnitude. Make sure you have runbooks for common problems and that your alerts contain links to related graphing and logging dashboards.
- **Resolution:** Manual resolution often results in additional problems, especially when remediation is applied under stressful conditions. If a recurring task can be documented, it can typically be automated. Ensure that you have automation tools, such as scripts, for common remediation actions (such as restarting a service and clearing log data). Additionally, it should be easy to automatically rollback to a previous release; switch to a failover cluster or DR (disaster recovery) environment; and fetch and restore backup data.

You are unlikely to automate 100 percent of your incident response lifecycle because some issues are just too complex. The good news, however, is that enterprises can automate the vast majority, thanks to modern technology and tools. The goal is to eliminate straightforward, repetitive tasks, enabling staff to focus on the complex issues.

Houston, We've Had a Problem

In the ITIL framework, *problem management* is the process responsible for the lifecycle of all problems. The goal is to eliminate recurring incidents, prevent problems, and minimize the effect of unpreventable incidents.

For example, if disk space on a production server fills up every week, a resolution step for such an *incident* would be to delete old data and free up space. But this solution is just a stopgap measure.

The *problem* is that the disk keeps filling up. A problem management resolution might involve automating the pruning of old data files, thus eliminating the problem.

Solutions to problems may be tricky to identify and implement, but they last longer and have a more positive effect on overall production health compared to incident management resolutions.

Identifying problems is hard to do manually. Most enterprises use support-ticketing tools, such as JIRA or ServiceNow, to record incidents. The power of tracking issues with tools such as these is that it enables the use of analytics to identify recurring incidents.

The challenge with this approach is that incident data is seldom recorded consistently and accurately. When IT incidents occur, staff is focused on service resolution — to the detriment of documentation, which is often done well afterwards. Even if the data is accurate, it's still hard to detect trends because manual human-documented information is typically unstructured and often incomplete.

Again, the solution here is automation. Alerting data should be processed automatically with a data analytics platform. Reports should rely directly on machine-generated, not manually entered, data. Furthermore, using an analytics platform designed with IT operations in mind will help you see your most problematic hosts and applications, and measure metrics such as incident volume and average time to resolution.

NOC, NOC. Who's There?

Perhaps no other term in the IT domain has been around as long as *NOC* (network operations center). Historically, data center health revolved around network health. When most people think about a NOC, they imagine a NASA-style room with tens or hundreds of staff staring at screens depicting network status.

In today's world, with agile development using rich application stacks built on top of virtual environments, public/private clouds, and containers, the traditional focus primarily on network management is outdated. Application bugs, database issues, and fixed resource consumption present just as real a risk for service outages.

Table 2-1 presents a few major shifts in the architecture of NOCs in the last 10 to 15 years.

Not surprisingly, NOC staff size tends to increase, even as service reliability decreases. The essence of service operation has changed, but the industry uses practices developed more than 30 years ago.

Table 2-1 Traditional NOC versus Modern NOC

	Traditional NOC	*Modern NOC*
Escalation Model	All alerts go through Level 1 and must be manually escalated to Levels 2 and 3.	Many alerts are routed directly to Level 2 or 3. Smaller organizations don't have 24/7 NOC rooms, instead relying on an on-call rotating Level-1 team.
Communication	Level-1 communications are issued via phone calls, email, and ticketing systems.	Communication may rely on mobile messaging solutions and social communication methods, such as chat.
Alert Volume	Several incidents occur each day.	Hundreds to thousands of alerts occur each day.
Tools	One major monitoring tool from a single vendor (BMC/IBM/HP/CA) and customized or proprietary dashboards.	Many different monitoring tools, heavily relying on SaaS (Software as a Service) and F/OSS (Free and Open-Source Software).
Stack Focus	Network, storage, and servers.	Users, applications, and databases.
Change Management	Changes occur during scheduled downtime windows; often weekly or monthly. NOC teams have clear visibility into changes.	Several changes per day, mostly automatically deployed. NOCs often discover changes after they occur.

With the adoption of SaaS-based tools, modern IT operations may consist of globally distributed teams. This renders traditional communications and methodologies ineffective. If companies want to regain control of their uptime while reducing costs, they need to drastically change the workflows in use in their NOC. Automation is at the heart of this transition.

Can't See the Forest for the Trees

Alert correlation is an important building block for automating both incident and problem management. One of the biggest challenges to automating actions triggered by alerts is the fact that any given incident is likely to generate multiple alerts.

For example, a high load on a large database cluster can result in hundreds of alerts. Data center-wide network issues, caused by hardware failures or DDOS (distributed denial of service) attacks, result in thousands of events. And an escalating error, starting with low memory, can gradually evolve into tens of additional alerts, including page faults, CPU errors, and disk IO errors.

A single alert offers little context on the effect or root cause of an incident. To understand what is happening, low-level alerts must be correlated into incidents to provide more context. For example, a single unreachable host might indicate that the server has crashed. However, multiple unreachable hosts in a subnet or physical location, such as a rack, are typically indicative of network issues.

In the first case, you might route the issue to a level-2 system administrator with the necessary Linux skills to resolve the problem. For the second case, it makes sense to route the issue to a networking team. But without automated correlation and classification, there's no way to automatically route the issue.

REMEMBER

Incidents and alerts often have a one-to-many relationship. That is, a single incident may result in multiple alerts.

When analyzing uncorrelated alerts, it is easy to assume that each relates to individual incidents. For example, an application may spawn numerous alerts, while in reality it is suffering from a much smaller number of incidents. Think of a disk failure on a critical server. Hosted applications would potentially fail, spawning many alerts of their own.

Correlating alerts into high-level incidents reduces the signal-to-noise ratio. Dealing with fewer incidents composed of multiple alerts is much more actionable for the support team.

Chapter 3

Correlation and Event Management

In This Chapter

- The past is always tense, the future perfect
- It is what you don't expect . . . that most needs looking for
- Bringing it all back home
- Chaos is just patterns we haven't recognized
- Always do whatever's next

We've been witness to shifts in data center architectures that have changed how outages affect organizations:

- Outages have a much bigger and more direct effect on business.
- It's much harder to evaluate the priority and determine the root cause of incidents.
- Humans can no longer scale to handle the volume of IT alerts they are faced with.

The solution lies in the junction of automation and data science. The complexity and scale of monitoring data call for an algorithmic solution to the problem. By removing labor-intensive manual operations, human beings can focus on the creative aspects of incident remediation. Tasks that require sifting through large volumes of data should be handled programmatically using algorithms, enabling organizations to significantly improve uptime while radically improving the efficiency of their operations workforce.

In this chapter, you find out how proper automation allows organizations to bring order to the chaotic world of alerting.

The Past Is Always Tense, the Future Perfect

With the number of IT alerts that the typical organization encounters increasing daily, there is a tendency to ignore lower priority issues by, say, increasing thresholds for alarming or filtering classes of alerts. The goal is to catch as many of the most important issues as possible while not getting distracted by the chaff.

> *"Our efficiency as an organization correlates most strongly with the quality of our alerts. In other words, our work queue defines what our scarce human resources work on in a given day. Given that, doesn't it make sense to supply that work queue with the highest quality, highest fidelity alerts possible to ensure that human resources spend their precious cycles on the highest value work? In other words,* ***more signal, less noise.****"*
>
> *– Joshua Goldfarb*
> *(Security Week 17 November 2014)*

REMEMBER

In operations management, the *signal-to-noise ratio* concept is defined as a measure that compares relevant or important alerts (the signal) to unimportant alerts (the noise).

We want to focus on the most imminent and severe events effectively and in a timely manner but avoid false positives. To not miss something catastrophic, however, we often have to live with a lower than desired signal-to-noise ratio, resulting in more false positives, which must be manually set aside. This larger than necessary volume of events prevents operational staff from spending as much time on each incident as they would like to, which may mean missing critical alerts buried in false positives.

In recent years, several monitoring solutions have attempted to address the alert noise problem. Rather than being built upon data science, these solutions implemented gross organizational mechanisms to reduce the noise in information. Unfortunately, this approach proved to be too simplistic to provide significant relief to operational staff.

Two of the most common strategies follow.

Aggregate metrics

In the *aggregate metrics* approach, several metrics are combined from a group of components. This approach configures alerts against that aggregate metric, rather than the measurements from an individual component.

For example, suppose that you're monitoring a cluster of 15 servers. Traditionally, you would measure CPU load on each server independently and configure an alert for each one. When CPU load on at least one server reached a certain threshold, an alert would be generated for it.

With aggregate metrics, you don't configure alerting on each individual CPU's load. Instead, you create a new aggregate metric measuring the *average* of CPU load on all servers associated with the cluster, as illustrated in Figure 3-1. You then configure alerts for that aggregated metric alone.

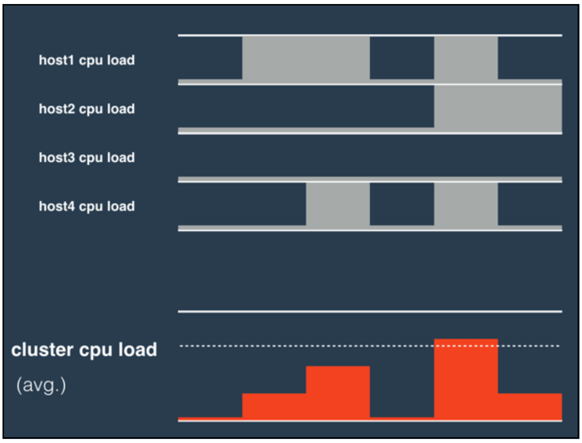


Figure 3-1: Visualization of an aggregate metric for CPU cluster monitoring.

It's easy to see why aggregate metrics reduce noise level:

- When a single server is experiencing load, the average remains low and you don't get an alert.
- When enough of the cluster experiences load to push the average above a threshold, you get an alert.

Aggregate metric-based alerting focuses on treating the potentially many components that provide a given service as a singular monitored entity. They accomplish this task by using a single combined alerting threshold.

The use of aggregate metrics for alerting might look like an elegant solution that reduces noise effectively. However, although this solution does reduce noise level, it also slows time-to-detection considerably. When an outage is beginning to evolve, the goal is to discover the outage as soon as possible. The sooner you discover the outage, the more likely you are to solve the issue before it affects customers.

With aggregate metrics, important symptoms are buried, thus postponing discovery of new issues. For example, high CPU on one server could be a good hint of what's ahead (such as a full cluster failure). But by averaging the CPU load on all servers, you will not get alerted on the failure of that CPU.

This situation is exacerbated as the aggregation groups increase. In a group of 2 CPUs, the previously described events would be detected fairly quickly. While aggregation groups minimize noise, they also reduce the weight of any singular incident in the cluster. Imagine how long it would take to detect the same imminent failure in a group of 50, 100, or 1000 CPUs.

Although aggregation of metrics is an effective technique for reducing alert noise, it also postpones the detection of important issues. As a result, few companies successfully piloted or chose to adopt this solution.

Hierarchical alerting

In the *hierarchical alerting* approach, an IT organization describes an inverted tree structure arranged by level of their infrastructure, as shown in Figure 3-2.

By identifying all the components involved in providing the billing service, you may group them and alert on an issue with the service on the failure of any given component.

Alerting is configured only on the topmost service. In this example, any alert generated by one of the components in this hierarchy would read: "Billing Service is affected: device X has a problem."

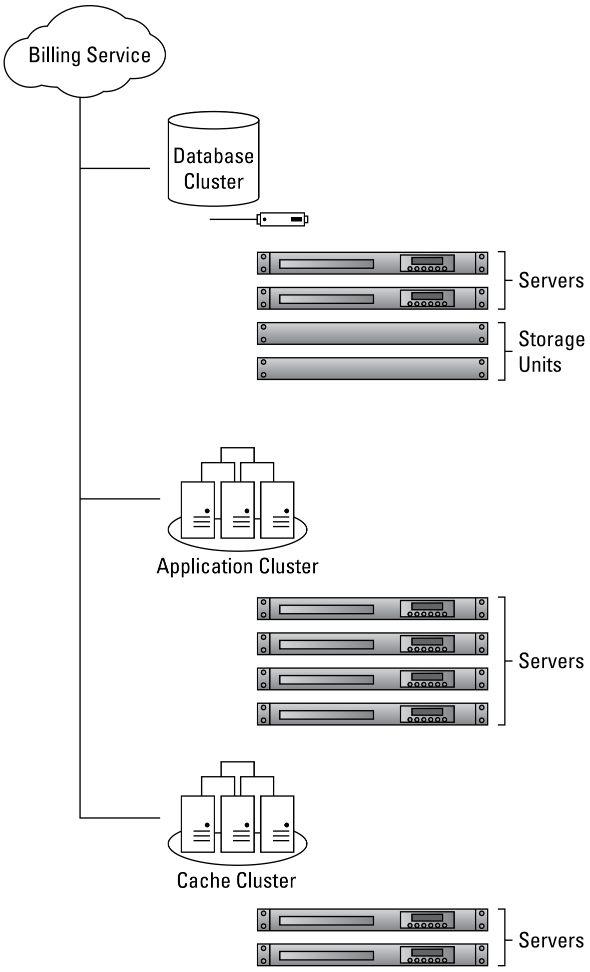


Figure 3-2: An example of a hierarchal alerting grouping.

At first glance, the hierarchical alerting approach has two attractive propositions:

- You can tie an issue directly to the affected business service.
- Huge alert storms are grouped into individual service-level alerts (thus reducing noise).

Sadly, this approach never succeeds in real-world environments. It is almost impossible to configure and maintain such a hierarchy in modern, dynamic data centers. They are in a constant state of flux, changing at high velocity; new applications and services are deployed all the time. By the time your team has finished defining this hierarchy, it's already out of date.

Further, modern environments are not hierarchical in nature and may contain multiple shared resources. The days of the monolithic application are long gone. The introduction of SOA (service-oriented architecture) led to modern microservices architectures. Applications are built in a graph-like manner, similar to a social network or a cluster of hyperlinked assets. For example, multiple applications might rely on the same database cluster and on each other. An analogy is presented in Figure 3-3.

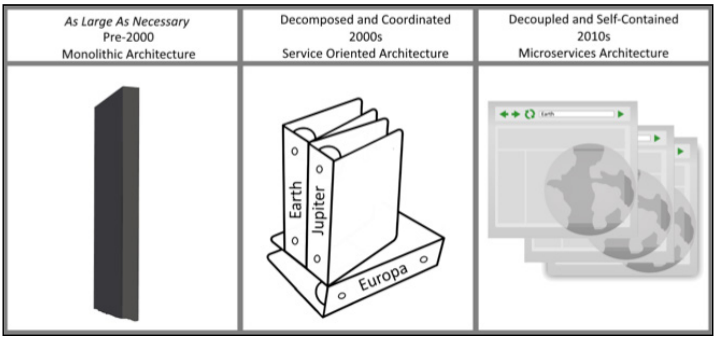

As Large As Necessary
Pre-2000
Monolithic Architecture
Decomposed and Coordinated
2000s
Service Oriented Architecture
Decoupled and Self-Contained
2010s
Microservices Architecture
Earth
Jupiter
Europa
Earth

Figure 3-3: Monolithic versus service-oriented versus microservices architecture.

Monolithic design patterns include all features in a single box. Any changes affect everything in that box, requiring careful coordination.

Non-critical alerts could create unjustified service-level alerts. For example, an unimportant "printer driver requires update" alert would propagate up to the service level. The most basic low-level issues are treated with the same level of severity as real outage-causing issues.

Hierarchical alerting has a great sales pitch: "Get a handful of service-level incidents instead of hundreds of low system-level alerts." But this panacea consistently fails to deliver in real-world scenarios.

In this context, a *design pattern* is an optimized solution to a commonly occurring problem in software design. A design pattern can also be thought of as a programming-language-independent template that may be reused based on collective experience.

The service-oriented approach decomposes and encapsulates components of an application as discrete services. These components are coordinated but loosely coupled, allowing them to be used in multiple contexts. Due to this decoupling, application state must be synchronized, making intercomponent messaging somewhat heavy.

The microservices model decouples components into discrete, independent application services. These services are self-contained and may be replaced at will, without fear of affecting other components.

It Is What You Don't Expect That Most Needs Looking For

We've painted a bleak picture so far, but all hope is not lost. A reliable solution exists for the problem of alert overload.

Alert correlation is an algorithmic approach to drastically improve the signal-to-noise ratio. An alert correlation algorithm identifies highly related events and groups them in real time.

Following are a few common examples of outages that can be correlated effectively using algorithms:

- **Failures of large database clusters:** Such failures result in hundreds of alerts, which the algorithm correlates into just one incident.
- **A data center–wide network issue caused by hardware failures or DDOS attacks:** The algorithm would cluster thousands of events into just a few incidents.
- **A gradually escalating error, starting with low memory and evolving into tens of additional alerts including page faults, high CPU utilization, and disk IO thrashing.** The algorithm correlates these alerts into one incident.

Algorithmic correlation typically results in a 75 percent to 99 percent reduction in noise for most environments. If your monitoring tools generate 6,365 alerts a day, for example, your team would need to process *fewer than a hundred incidents* in the same timeframe.

Correlation also helps with RCA (root cause analysis). By grouping all related symptoms, the detective work becomes considerably simpler. For example, a single "application is slow" alert provides little insight, but an incident grouping of an "application is slow" alert with a "low memory alert on the application's server" alert is another story.

Alert correlation provides great relief in the form of noise suppression as well as simplifying incident investigation and root cause analysis. Programmatic correlation provides advantages in time to detection, while easily supporting modern environments and design patterns at significantly more reasonable costs than the use of aggregate metric-based or hierarchal alerting.

Table 3-1 presents a summary comparison of aggregate metrics, hierarchal alerting, and alert correlation.

Table 3-1 Aggregate Metrics versus Hierarchal Alerting versus Alert Correlation

	Aggregate Metrics	*Hierarchical Alerting*	*Alert Correlation*
Time to detection	Delayed	Immediate, but might overemphasize unimportant alerts	Immediate
Root cause analysis	No value	Somewhat helpful	Very helpful
Support for modern environments	Handles dynamic clusters well, but doesn't support complex dependencies	Doesn't support microservices architectures and weak support for dynamic clusters	Algorithmic approach can detect complex relationships in highly dynamic environments
Maintenance cost	Average cost	High cost	Low cost

Alert correlation provides a reliable and pragmatic solution to the operational problems arising from the complexity and scale of modern data centers.

Bringing It All Back Home

At this point, you must be asking yourself, "What kind of magic is this alert correlation stuff?" The world of IT operations values *transparency* and *predictability* in solutions, so we'll explore how alert correlation algorithms work, what data points they employ, and what it takes to use them in your environment.

First we need to discuss two important concepts: heuristics and clustering.

Heuristic algorithms are methods that prioritize practicality over theoretical perfection. They focus on finding the best possible solution quickly and easily.

Many algorithms work well in theoretical models and simulations but fail in real-world scenarios. Why? Because data quality (the relative utility of information for operations, decisions, and planning) is highly variable and subject to external constraints.

By contrast, a heuristic algorithm acknowledges that the world is full of imperfect information and unexpected constraints. A heuristic for a navigation system might generate a route that prefers frequently travelled roads, even if additional optimization and fine-tuning could have resulted in a two-minute gain.

As you'll see, heuristics play a central role in alert correlation.

Clustering is a family of algorithms in which the goal is to group a large sample set into a limited number of related objects. Google News (news.google.com) employs clustering to categorize items from disparate sources based on story. In marketing, analytical tools are used to identify distinct clusters of customers based on shared interests. That's why the data we freely give to social media companies boosts their market capitalization, giving them price-to-sales ratios of 10 to 15 times and price-to-earnings ratios in excess of 50 times, often when there is a lack of positive earnings.

Figure 3-4 provides a visualization of the output of a clustering algorithm. The algorithm input is a set of points. The algorithm output consists of clusters of points based on geometrical affinity. You can see the different clusters visualized using colors.

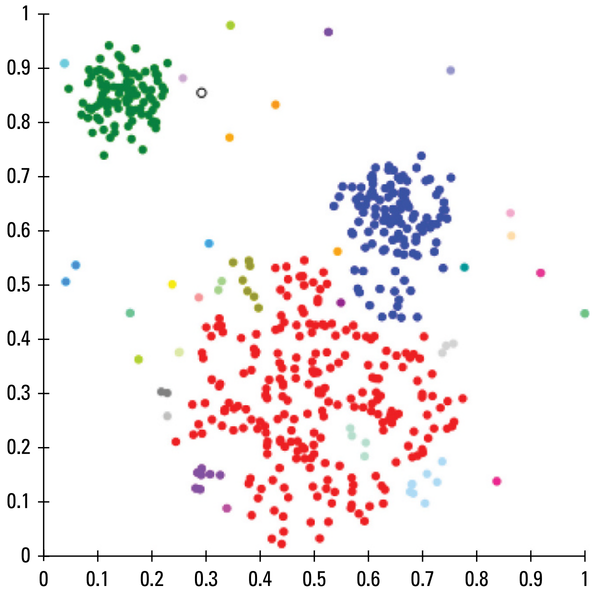


Figure 3-4: Visualization of the output of a clustering algorithm.

It's easier to see the relationship between data points when presented in obvious groupings.

Alert correlation is a form of clustering algorithm. Its input is a set of alerts generated by monitoring tools. The output are incidents in which highly related alerts are grouped.

How Does Alert Correlation Work?

Alert correlation uses multiple heuristics to calculate the relationship between every set of alerts and then clusters alerts into incidents if the alerts are highly related.

Alert correlation may employ different kinds of heuristics. The most typical ones in use follow:

- **Time distribution:** Alerts occurring within a short timeframe are more likely to be related than alerts uniformly distributed over time. For example, if no Disk IO alerts were fired between 5 a.m. and 6 a.m., and then 20 distinct Disk IO alerts were fired within a 5-minute time window, all 20 alerts are probably related.
- **Statistical patterns:** A relationship between alerts is inferred by sifting through historical examples. Consider the case that over time an algorithm detects that whenever application X experiences high latency, host Y experiences high CPU load. We can then infer that the two symptoms are related, and use that knowledge for future correlations.
- **Topological relationship:** The way that an environment is structured has a huge effect on correlation. Alerts coming from the same database cluster are much more likely to be related to one another than alerts coming from the same data center. Devices attached to the same switch are more likely to fire related alerts than devices attached to separate switches.
- **Foundational relationship:** Some alert types are, by definition, more likely to be related than others. For example, it is easy to see the implied connection between page faults and low memory alerts. By contrast, low disk space is less directly tied to high application latency.
- **Reinforcement learning:** The algorithm learns based on user feedback. Given two correlated alerts, the user can indicate that the correlation is incorrect. The system avoids repeating the same mistake again. Similarly, the user can indicate that two uncorrelated alerts are related. Again, the system learns this behavior, and correlates these alerts the next time they occur.

Not all correlation algorithms are equal. They may use different subsets and combinations of heuristics and assign different levels of importance to those heuristics.

You can measure the utility of an algorithm in several ways:

- **Efficiency:** Does the algorithm provide meaningful noise reduction?
- **Accuracy:** Are correlations correct? Are false positives and negatives minimized?
- **Practicality:** How easy is it to collect the information the algorithm requires to function?
- **Time to value:** How long does it take for the algorithm to start generating measurable value?

Another important dimension of alert correlation algorithms is their temporal nature. Most clustering algorithms are used offline, meaning that they need to see all their inputs before they can detect clusters.

Alert correlation, however, is time sensitive. The value of alert correlation decreases as the time to notification increases. The expectation is that correlations be provided in real-time. In a perfect world, you would expect to be notified instantly when the first alert occurs. You don't want to be notified five hours later, after the algorithm has consumed all the related alerts.

This sensitivity to delay leaves correlation algorithms with the challenge of operating efficiently in real-time, managing the trade-off between accuracy and timeliness and continuously updating their correlations as outages evolve.

Chaos Is Just Patterns We Haven't Recognized

The reason we configure alerts is to give us an indication when something is broken. The problem with alerts, however, is that they indicate only *when* something is broken. More often than not, they don't tell you *what* is broken or *why.* For example, a Slow Database Query alert indicates a problem.

What applications are affected?

How are they affected?

Why is it slow to begin with?

Is it a network issue?

A load issue on the database cluster?

Alert correlation not only reduces noise, enabling you to identify the *when,* but also provides insights into the *what* and the *why*.

Consider the visualization of an incident presented in Figure 3-5.

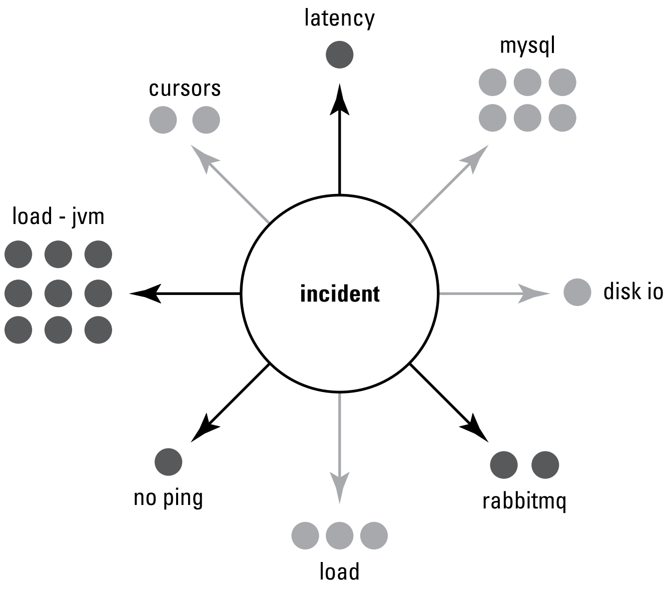


Figure 3-5: Visualization of an incident and its component alerts.

Each circle represents an individual alert.

You can see six alerts related to MySQL, nine alerts related to JVM-load, two database cursor alerts, and more. In isolation, any alert provides little information. After they are all correlated into one incident, however, the picture becomes much clearer.

Continuing with the example, in terms of effect, we can immediately conclude that the entire MySQL cluster is affected, and we can see that a big part of our Java infrastructure is experiencing load. We can also see that a specific application has latency issues.

In terms of root cause, we now have a clear list of our prime suspects. We see all related symptoms in one context, and we can treat each of them as a clue. In this particular case, it appears that the application is under heavy traffic, resulting in across-the-board load in multiple services.

Getting to the same level of insight without correlation would take hours.

Always Do Whatever's Next

So far, you've seen how alert correlation can aid in both reducing noise levels and turning alerts into insights. One additional area worth discussing is how alert correlation improves outage-related communication in an organization. During outages, communications fall by the wayside.

Finger pointing

Finger pointing occurs when we lack clarity around the effect and cause of an incident.

For example, the NOC escalates an issue to the application developer, per their runbook. The developer redirects the NOC to the network and virtualization teams. The network team, in turn, insists the issue is application related, while the virtualization guy suspects a storage issue.

The end result: a lot of finger pointing and little progress in restoring service.

Broken information flow

We've all been there. An issue is routed to a certain team or individual but crucial information is missing.

For example, the NOC sees a latency issue and communicates it to the developer. The developer is not aware that a network issue is presenting alerts at the same time, and focuses on potential inefficiencies in their code rather than interfaces to the network.

Incorrect prioritization

A sense of urgency is hard to communicate accurately in writing. Often, a critical problem can appear like a minor issue in an email message. Alternatively, a low-priority issue could be perceived as highly urgent, forcing an engineer to context switch from another important task.

Lost in translation

Solving communication issues is not easy. To address them you need a combination of tools, processes, soft skills, and perhaps most importantly, corporate culture.

Dealing with communications issues is a never-ending struggle. You are unlikely to reach an equilibrium in which communication is perfect. In fact, an entire book could be written just on the topic of communication during outages.

The good news is that alert correlation can provide a first step towards improvement.

Alert correlation enables you to retire the concept of an alert from your company's communication channels. An IT alert is merely a symptom and should never be prioritized, routed, or recorded in isolation.

With alert correlation, a new concept comes to life: The incident. The *incident* represents a collection of all related symptoms treated as a singular unit. To communicate any outage, you always reference the incident. This approach guarantees healthy information flow. All recipients receive fully contextual information instead of an isolated alert.

Further, the adoption of the concept of incidents enables us to record not only machine data (for example, alerts) but also human interactions.

Whenever someone collaborates around an incident (adding notes, routing and rerouting the incidents, prioritizing it, and so on), this human-generated information needs to be recorded as part of the incident. Now, for the first time, we can seamlessly marry machine data with human-generated information in one place. This approach wouldn't be possible without alert correlation.

For example, without alert correlation, the ops engineer might add a note to a CPU load alert, while the app developer would add another note to an alert representing a log exception. Of course, this situation could have been avoided if all related alerts were grouped in one place, identified as the incident. The developer and the ops engineer would have collaborated on this incident.

Companies that use alert correlation often implement automated escalation to enable better collaboration. For example, any incident including database-related alerts can automatically spawn JIRA tickets in the DBA queue. No need to worry about missing context — it's all in the ticket, thanks to alert correlation.

Another example would be companies consuming incidents directly from a ServiceNow table. Automated escalation is impossible if every alert becomes a ticket (representing thousands of tickets per day). With alert correlation, only grouped incidents are turned into ServiceNow tickets.

In the end, alert correlation will help you improve communications during outages, which in turn will reduce your time to resolution.

Chapter 4

Ten Things to Remember

In This Chapter

- The data center has changed
- We live in a golden age of monitoring
- Noisy alerting is a major business issue
- Noisy alerting is not a fact of life
- Putting more people on a problem is a Band-Aid
- Data problems require data science
- Automation goes beyond orchestration
- Correlation reduces downtime
- Correlation boosts collaboration
- It's primetime for alert correlation

Here we are at the end of the book and you're probably asking yourself, "How am I going to remember all this?" We have you covered. If you walk away with nothing else, here are ten takeaways that you can use the next time you find yourself discussing the current state of data center monitoring.

The Data Center Has Changed

Any sufficiently advanced technology is indistinguishable from magic.

Clarke's Third Law, Arthur C. Clarke

Today's data centers would sound like science fiction to IT professionals in the 1990s: millions of physical machines running hundreds of millions of emulated operating systems supporting billions of constantly changing applications and services.

This is the new reality in which we live, and data center growth is not slowing down — it's accelerating. IT organizations need to embrace this rate of change if they want to remain successful in the coming decade.

We Live in a Golden Age of Monitoring

The power of visibility can never be underestimated.

Margaret Cho

Visibility around infrastructure health is no longer a bottleneck. There is a robust market of monitoring tool offerings. We have tools that excel at monitoring devices, applications, cloud instances, users, containers, and practically anything else you can think of. Open-source, commercial, SaaS, and on-premises solutions are available.

In fact, we have so much visibility today that the new bottleneck is a direct result of all the data generated by those monitoring tools.

Noisy Alerting Is a Major Business Issue

Why then, can one desire too much of a good thing?

Shakespeare, As You Like It

Too many alerts generated by monitoring platforms affect many business-critical aspects of your organization. Critical outages are noticed too late, resulting in expensive downtime. Your headcount requirements (and with them, the staffing costs) are going through the roof. Productivity is plummeting and frustration is skyrocketing.

And you are fighting to keep your best and brightest in the face of employee churn.

Noisy Alerting Is Not a Fact of Life

One day everything will be well, that is our hope.
Everything's fine today, that is our illusion.

Voltaire

We've become so accustomed to that inbox full of alerts every morning that many of us have just given up. We just assume that alert noise is a part of the IT operations lifestyle.

But if we can succeed in not just treading water but getting out from underwater, we'll see that's not the case. Companies willing to invest in resolving the problem understand that alerting noise can be significantly reduced. In fact, companies that have taken action have been able to reduce their noise levels a hundredfold and generate operational awareness that was simply not available previously.

Putting More People on a Problem Is a Band-Aid

Too many cooks spoil the broth.

Universal proverb

Granted, sometimes hiring a few more operators or engineers is the fastest way to scale if you need to immediately handle growing alert volumes.

However, staffing is a short-term solution. Alert volumes will continue to grow, and if you rely on growing headcount, soon enough your NOC will include more operators accomplishing less and feeling frustrated.

Data Problems Require Data Science

To a man with only a hammer, a screw is a defective nail.

Orson Scott Card

The volume of data generated by today's operational environments is enormous. We cannot rely on manual workflows if we want to make intelligent decisions.

Algorithmic solutions are the key to making fast, accurate, data-driven decisions. From marketing to finance to IT, the world is leveraging algorithms to handle the growth in modern data volumes.

Automation Goes beyond Orchestration

The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency.

Bill Gates

We all employ automation for provisioning servers or deploying code. But more and more companies are realizing that's only half the picture.

What about incident management workflows? Prioritization, correlation, investigation, escalation, and remediation — these are all automatable.

Correlation Reduces Downtime

There's no such thing as downtime for your brain.

Jeffrey Kluger

Important symptoms can easily fall through the cracks when you're flooded with alerts, resulting in late detection and difficulty in triaging and performing root cause analysis of issues.

Grouping related alerts together allows operators to quickly identify the source of an outage and assess its effect. Correlating alerts provides visibility on how incidents evolve and accelerates root cause analysis.

Correlation Boosts Collaboration

Every collaboration helps you grow.

Brian Eno

Alerts are mere symptoms generated by monitoring software. The process of grouping related alerts into incidents facilitates human workflows. Alert correlation bridges these two types of information by grouping together alerts into high-level actionable incidents. The result is better information flow in your organization and less finger pointing.

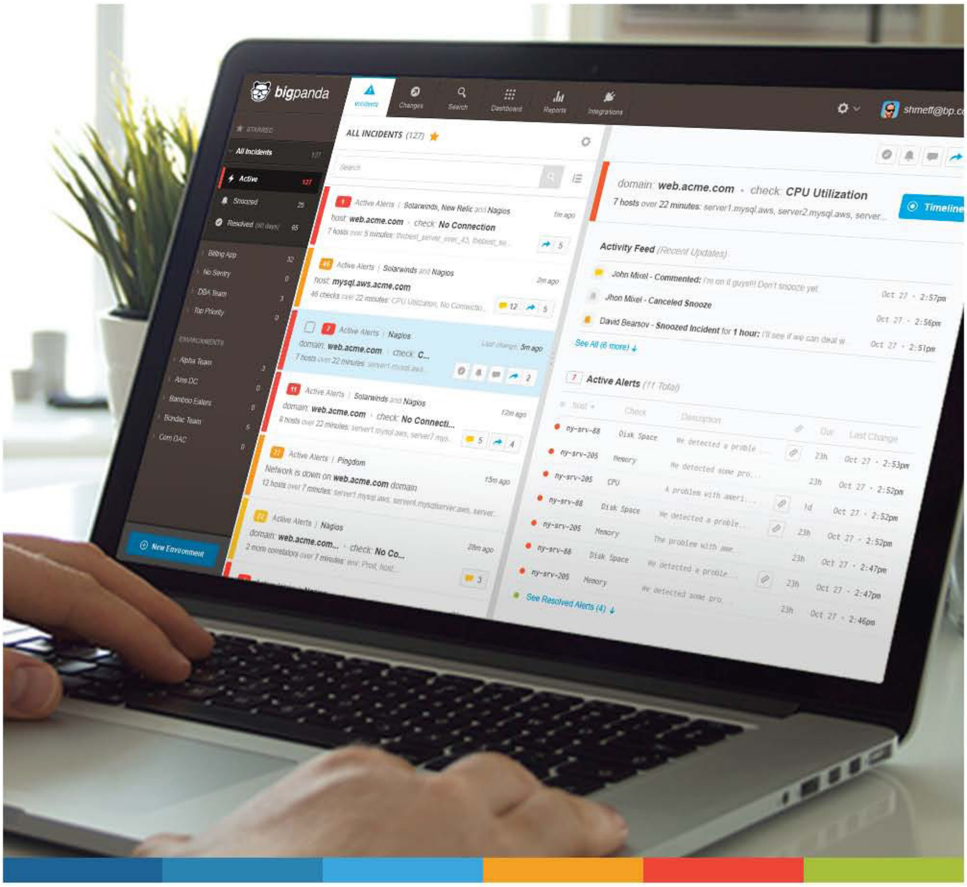
It's Primetime for Alert Correlation

The future is still so much bigger than the past.

Tim Berners-Lee

When people talk about algorithmic solutions to difficult problems, it's easy to dismiss this as snake oil. But it's important to grasp that the world has progressed a great deal in the last decade, and algorithms are indeed capable of solving extremely difficult problems.

Computer algorithms trade on Wall Street; they automatically tag our friends in Facebook pictures; and they even drive cars on American highways. Algorithmic alert correlation is not only a practical solution but also quickly becoming the industry standard.



Eliminate alert noise.

- Automatically correlate alerts up to 99%
- Reduce MTTR by 85% on average
- Integrate with ServiceNow, JIRA, and more

bigpanda | Visit **bigpanda.io** to get started

Learn how to reduce alert noise and spot critical issues faster

IT systems are more diverse, complex, and agile than ever before. As a result the amount of alerts that operations teams have to deal with has increased by orders of magnitude. Alert correlation is designed to reduce the strain and boost productivity by transforming high volumes of machine-generated alerts into actionable insights.

- ***Alert correlation basics* — learn how alert correlation works and how it can help to more effectively manage your services**
- ***Alerts in context* — understand how correlating IT alerts into high-level incidents help you improve detection, accelerate remediation, and increase productivity**
- ***Deployment* — review the drawbacks and limitations of legacy approaches to reducing alert noise**
- ***Business value* — understand why alert correlation matters and how it can benefit your bottom line**

Open the book and find:

- **Challenges that automated alert correlation can help you overcome**
- **Tips for improving the efficiency of your ops team**
- **Performance advantages of algorithmic data clustering and normalization**
- **Limitations of legacy approaches to alert correlation**
- **How IT alert overload affects your business**
- **Why all correlation solutions are not created equal**

Allan Konar is Vice President of Solutions Architecture for CaféX, a leader in WebRTC innovation, enabling business applications with real-time communications capabilities. He has more than 20 years of hands-on security and performance management experience spanning application, web-based, wireless, and telecom industries.

ISBN: 978-1-119-27849-8
Not for resale