Making Everything Easier!™

Oracle Special Edition

In-Memory Data Grids FOR DUMMIES®

A Wiley Brand

Learn to:

- Scale quickly to meet customer demand
- Save costs by offloading shared services
- Increase satisfaction via fast, fault-tolerant infrastructure

Brought to you by

ORACLE®

Michael Wessler, OCP & CISSP

In-Memory Data Grids

FOR DUMMIES®

A Wiley Brand

Oracle Special Edition

by Michael Wessler, OCP & CISSP

FOR DUMMIES®

A Wiley Brand

In-Memory Data Grids For Dummies®, Oracle Special Edition

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

ISBN 978-1-118-92141-8 (pbk); ISBN 978-1-118-92310-8 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

Senior Project Editor: Zoë Wykes

Business Development Representative: Karen L. Hattan

Project Coordinator: Melissa Cossell

Special Help from Oracle: Craig Blitz, Jens Eckels, Ayalla Goldschmidt, Mike Lehmann, Cameron Purdy

Table of Contents

Introduction

All businesses know that promoting a positive user experience is critical to keeping your customers happy and attracting new ones. How can IT support an environment that is responsive and reliable even under spikes in user demand? One proven solution is In-Memory Data Grids (IMDGs). Leveraging IMDGs provides the speed, reliability, and scaling required — yet often lacking — in so many computer systems.

About This Book

In-Memory Data Grids For Dummies, Oracle Special Edition, consists of five short chapters that identify the requirements to be successful in a dynamic IT world and to give customers the user experience they demand. This book explores how IMDGs provide faster and more stable service to customers while solving difficult scaling and reliability challenges, and gives a clear path to improving the customer experience with IMDGs.

Icons Used in This Book

In this book, you occasionally see icons that call attention to important information. Here's what to expect.

This icon points out information that you'll want to remember over the next few months, years, or until retirement!

You won't see a lot of highly technical information because I don't want to give you a headache, but you will see this icon a few times.

This denotes information that may make your life a little easier if you follow the advice. Odds are this information is included because I didn't heed it myself at one point!

Learning from someone else's mistakes is better than making them yourself. This information alerts you to things to watch out for or areas that could cause problems.

Beyond the Book

Technology is always evolving, so in order to stay up to date on the latest in Oracle In-Memory Data Grids products, here's a website to visit: `http://www.oracle.com/us/products/middleware/cloud-app-foundation/coherence/overview/index.html`.

Where to Go from Here

This book isn't big enough to include all the information there is about IMDGs, but I can give you the most important information to get started with. You can read the book from beginning to end (it shouldn't take too long), or you can just jump to whatever chapter interests you most. Either way, please enjoy!

Chapter 1

Identifying Market Drivers

In This Chapter

- Understanding changing market factors
- Defining today's tech trends
- Discovering why scaling is important
- Building scalable systems that offload shared services

Change is a constant part of business and has a direct impact on the technology used to support business. As data grows and the demand for instant access to complex business processes increases, the supporting technology must evolve to keep up. Businesses that effectively meet these challenges will reap great benefits while businesses that don't will suffer.

This chapter examines the changing business market in terms of how industry trends — such as mobile, cloud, and the growth in the number of connected devices — are driving demand for faster, more scalable computer systems. It also explores why businesses must use fast and scalable computer systems to provide the data and processing needed for a modern, technically savvy, and impatient consumer.

4

Understanding What's Driving the Market

Today's consumers are a very demanding and impatient group; they expect a seamless experience from any website or computer system that they use. As technology has become easier to use and more widespread, customers' expectations have increased. These days, users expect the following:

- Constant web access, social media, and mobile device integration
- Websites or applications that are quick, responsive, and always available
- An intuitive, friendly, and feature-rich user interface (UI)
- Real-time responsiveness and seamless integration of applications
- Immediate access to new products and services as they become available

Although customer expectations and demands are technologically challenging to meet, they must be met. If a website is slow or a sales application is cumbersome, previously loyal customers won't hesitate to move to a competitor in a matter of seconds. By itself, technology won't make a poor business idea successful, but without effective technology, many good business ideas will not meet their fullest potential.

Architecting for Today's Challenges

Information Technology (IT) is a continually evolving industry, often shaped and sometimes radically redirected by industry trends. Trends often build on each other and offshoots are common. As the Internet emerged, it became a place to work, conduct commerce, and entertain. Mobile devices brought the power of the Internet to the everyday person with near around-the-clock access. One could argue that the impact of the Internet is one of the greatest societal changes in a century. However, this change didn't occur overnight, and it took considerable architectural design and infrastructure to make this trend a reality.

Today, devices of all sizes, ranging from credit cards to household appliances and even to our homes and cars, are being electronically integrated with each other and the Internet.

Existing technology that is gaining popularity, such as mobile devices, social media, cloud computing, and always-connected devices, requires the management of large volumes of data from an ever-expanding universe of highly varied data sources.

Supporting these consumer and device-to-device (often termed the *Internet of Things*) trends requires a robust IT architecture that supports the following:

- **Availability:** Infrastructure-supporting devices must always be available and ready to receive and process large amounts of data.

- **Connectivity:** Devices need to reliably and quickly connect to the network when data is transferred.
- **Performance:** Devices and their communications must be fast and efficient; delays and bottlenecks are amplified and customers won't tolerate them.
- **Security:** Devices interacting with every facet of users' lives and commerce must be secure or users won't adopt them.
- **Scalability:** Devices will soon number in the billions; the supporting infrastructure must support these vast numbers and be able to process events from them in real time.

The architecture to support growing use by people and their devices is taking center stage for IT vendors and becoming part of the infrastructure required by everyone.

Why You Want to Scale Applications

Nothing is more frustrating to customers than trying to use an application or make a purchase only to see that application move slowly, freeze, time-out, and ultimately fail.

Customers have been conditioned to expect quick response time, so if an application is slow, the customer may leave the site. For example, during an online retail sale, customers may abandon their shopping carts if the site freezes up. Very often, a system is slow because it wasn't architected to scale its application tier to meet user demand.

Scaling for success

A harsh lesson in failing to scale is encountered every year by companies advertising during the Super Bowl, the championship game of American-style football's National Football League (NFL). As you may know, the Super Bowl is known for offering humorous, clever, and compelling advertising during its airing. Many people even say they watch the Super Bowl solely for the commercials. Because of this, large companies spend millions of dollars on flashy Super Bowl campaigns that attract viewers to their websites or other properties, only to find that their computer infrastructure collapses under the weight of the millions of visitors. Many otherwise successful Super Bowl commercials have failed to achieve the desired sales results because the company's computer infrastructure couldn't support the short-lived spike in web traffic. Further, the social media backlash regarding site failures and application crashes is immediate, harming the company's brand. The window of time to capitalize on a Super Bowl ad campaign (like many other campaigns) is relatively short. Lacking the capability to dynamically meet the increased user demand can impact immediate sales and longer-lasting brand impressions.

Understanding the urgent need

As IT departments are faced with the problem of growing user demands and their own shrinking budgets, they can find themselves facing certain challenges when building new systems or supporting existing systems:

- Computer systems use shared infrastructure to ensure that stable and trusted components are used within the enterprise.

- Application usage and workload are often difficult to forecast accurately at the system's inception.
- Unpredictable usage spikes and new or changing requirements are difficult to plan for in advance with enough lead time to take corrective action.
- Performance problems within highly specialized components or databases or infrastructure can be expensive and time-consuming to correct.

Fortunately, a solution exists where IT can meet the increasing workload demands while managing costs. This solution is called *scaling.* I explore scaling more fully in subsequent chapters.

Dynamic, Real-Time Systems

Successful systems need to scale rapidly to meet new and unexpected business needs. Systems should be designed to scale linearly and dynamically to take advantage of changing requirements and surging demands. A well-designed system will scale to

- Provide virtually unlimited processing power and memory as additional cluster members are added.
- Expand in real time and dynamically to meet increasing requirements.
- Increase capacity in a linear and predictable manner.
- Leverage commodity or integrated systems that are easily added without complexity.

Modern technology supports horizontal scaling by adding additional nodes "on the fly." New nodes join the system dynamically and begin accepting workload

as soon as they are initialized. This real-time scaling capability shields users from negative experiences due to gradual increases in workload and processing spikes.

Offloading Shared Services

Within many systems, retrieving data involves going to distant, legacy, or undersized mainframes, databases, or external data sources. Obviously, data must be stored at rest somewhere (the data tier), but retrieving data for every request from a performance and processing standpoint is expensive in time and hard cost. Furthermore, many of these data stores cannot scale rapidly enough to meet new workloads nor can they be updated rapidly to reflect changing business requirements.

System architects design computer systems to leverage application-tier scaling and incorporate caching to *offload* the pertinent data from the data tier to reside at the application tier. This design relates to the best practice of keeping *live data* near the user while reducing the processing requirements on the data tier. Live data is the high-usage data that is accessed frequently rather than historical and seldom-used data that is stored deep within back-end data stores and databases.

The capability to rapidly provide this live data to the user is what makes a scalable system fast. For example, many applications depend on shared or partner services that may not be dedicated to the application. A particular application might use someone's social media network preferences to display a certain type of information or pull up recommendations. Rather than retrieve that network's data over and over, it makes sense to keep partner data in memory for fast access. This improves the user experience and reduces dependency on the partner system.

10

Chapter 2

A Peek at Data Grid Use Cases and How They Work

In This Chapter

- Benefiting by offloading shared services
- Scaling applications to handle more users
- Providing real-time data to customers

Understanding the key use cases around data grids is necessary for entire technical teams ranging from developers and architects to the systems managers in charge of the resulting applications. When people understand a technology's sweet spots, they're more likely to effectively implement that technology to its fullest potential.

This chapter explores key use cases for IMDGs and takes a peek under the covers of how they operate.

12

Evolving Challenges of Scalability

Scaling applications refers to the ways that companies grow and shrink to meet an increased load, be it from more users, more data, or new use cases that require more processing. A system should scale dynamically so that it can always meet current demands, yet not sit idly when demand shrinks. An IMDG is *horizontally* scalable, meaning that it scales by adding compute resources.

By contrast, a *vertically* scalable system limits headroom. In the Super Bowl example from Chapter 1, where a company's sudden exposure creates a demand on its website that it can't meet, a vertically scalable design sets maximum limits to the number of customers that one system can handle. A horizontal design allows you to scale beyond that single server. Horizontal design should consider how to scale all resources, including compute, network, and memory, so they don't become bottlenecks. Today, web, mobile, and cloud applications — as well as the young Internet of Things phenomenon — are forcing companies to address how they handle the increasing volume of application users, and scaling to process data from those users.

Data centers must now be available and reliable 24/7, and further growth must be dynamic and automated so that customers are guaranteed high performance.

IMDG products are designed to alleviate these challenges. Further, IMDG products such as Oracle Coherence can provide multi-data-center guarantees for constant availability of critical applications.

Understanding Offloading

Offloading is a term that reflects the removal of strain from back-end databases, mainframes, and shared or partner services. The growth of an application (due to an increased user base, transactional volume, or data volume) can cause strain on existing systems.

An example of offloading is when an application originally designed to get a certain piece of data directly from a database (such as a hotel room quote from a single vendor) strains the database connection because a new user group — such as a new influx of mobile users or a new geography being served — begins using the application over and over to access that same data. This situation not only slows the results of the database request, but also strains the infrastructure that the data passes through, such as a shared web service or a piece of the application that draws on external data.

IMDG technologies (such as Oracle Coherence) cache frequently accessed data so that the application doesn't need to go all the way back to the database for every request. The use of offloading also reduces dependencies on external shared services — in the example, an aggregating provider of hotel quotes— so that if a shared service is down, its cached data is still available.

Offloading in Action

IMDGs provide a rich set of functionality to offload shared services. Most IMDGs offer integration with databases and other data sources so that access to the back-end data source is transparent to an application. The application reads the data, and the cache loads the data from the data source if it isn't present. Similarly,

with writes to the cache, IMDGs write data back (asynchronously or synchronously) to the data source, further offloading by batching writes.

All IMDG products allow you to control expiration and eviction of cached data. Some IMDG products give you fine-gain control of these options by allowing you to refresh ahead before your data expires.

By automatically partitioning data across a set of servers, IMDG solutions allow full horizontal scalability. Advanced IMDG solutions provide the ability to dynamically add and remove servers to scale up and down on demand. In these advanced solutions, backups are stored on disparate servers, allowing for *high availability* (HA) if an individual server or data center crashes. These HA schemes also prevent workload storms on your back-end data sources in the event of server failure.

Scaling the Application Tier

Strains on shared services caused by scaling do occur and must be addressed. Scaling also causes problems at the application tier. As more users are added to a system, applications are forced to manage more user data (such as HTTP session data during an online shopping experience) and more transactions. Applications also have to manage a growing volume of application data in addition to user data.

Application servers have several techniques to handle this scalability challenge, but eventually the complexity of scaling the application tier and managing the data becomes too complex without a purpose-built solution.

Leveraging Application Tier Scaling

Many of the features used to offload shared services are applicable across all IMDG solutions. The most obvious challenges that architects face include how to store and manage a growing amount of data on the application tier without incurring response-time delays (for example, because of Java Garbage Collection), how to keep the data in a growing number of application instances in sync, and how to avoid each application instance from making its own calls to shared services. By providing a virtual data layer, IMDGs make it easy to access data without having to know where it resides. This means that applications offload the responsibility of keeping data caches in sync and virtualizing access to shared services. Because the data grid is completely scalable, access times to data remain constant as the data grows.

For clients that make repeated calls to the cache for the same piece of data, IMDGs allow you to configure a near cache in the client application. A *near cache* combines the speed of local data access with a fully scalable distributed cache. IMDGs manage the task of keeping near caches in sync with the distributed caches that sit behind them. For example, consider a large set of cached read-mostly reference data (maybe a map of product SKU to descriptions). A small subset of this reference data may be used quite heavily. The reference data would reside in a distributed cache for scalability, but the application clients would declare a near cache so that frequently accessed reference data would be

accessible locally. Additionally, many IMDG products integrate with popular application servers to allow you to automatically offload HTTP session-state data. By offloading session-state data, an application-server instance removes response-time jitter associated with Java Garbage Collection. This improves performance.

Exploring Real-Time Data

Real-time data is defined by its very name as the ability to process large amounts of data at real-time speeds. Although not every application has this requirement, many industries rely on the quick processing of data feeds as a competitive advantage.

Financial services companies that need to provide real-time prices and risk information to their traders and customers exemplify real-time data. Outdated information may negatively impact profits. Calculating price and risk information is more than a matter of moving one piece of data from one place to another; instead, complex calculations may need to be performed based on an incoming stream of data (in this case, on market events) and sent downstream to trading applications in real time.

In another example, the marketplace offers multiple wearable fitness devices that tie into mobile applications to track exercise patterns and workout routines. Many of these devices calculate multiple data points simultaneously, such as your heart rate, distance traveled, route taken, and GPS position. This data is streamed back to a data center where an application further processes and enriches the data, for example by comparing it to previous workouts or those of a group of peers and then sending those results to a downstream application to

update social sites or leader boards in real time. Because of the large volume of streaming calculations that must be performed, a scalable platform that can perform calculations in parallel is needed.

So, even though caching data in the data grid can increase speed and improve scalability, processing a massive amount of real-time transactions demands more capabilities of IMDGs than simple caching.

Accessing Real-Time Data in a High-Speed World

Using a simple bank balance problem as a traditional example, assume that you've managed to speed up access to data by caching customer balance information. At this point, a more traditional architecture would work something like this:

1. The application server receives an event saying it's time to perform a transaction, such as a customer who wants to withdraw money from a bank account.
2. The application server goes to the cache to retrieve the balance and any other information needed to perform the transaction, such as placing a lock on the data.
3. The application server then updates the bank balance and writes the data back to the cache (and the cache writes the data back to the database).
4. The application server releases the initial data lock and lets the user know that the transaction is complete.

That's a lot of work for one transaction!

Using IMDGs, this same banking transaction would be performed like this:

1. The application server receives the event notification that a bank withdrawal is being made and calls into the data grid to process the transaction using what is called an entryProcessor.
2. The transaction is completed, and the user is notified. No locks are necessary.

Using the data grid entry processor allows the grid to update the balance in place — and return a result, including an indication of whether the transaction succeeded.

Furthermore, you're no longer moving data around to process it. This example is about moving bank account balances, which are small data types, but consider the benefit if you were moving large documents or files instead. By moving processing off your application or web tier, you create a more scalable processing tier and allow the application and web tier to serve customers and manage customer interactions much faster.

IMDGs have the ability to "listen" to cached objects when they change. By doing so, you can build scalable systems that respond to state changes and automatically trigger new calculations or processes to be performed.

Finally, continuing with the banking example, a process running on the data grid may listen to balance updates and trigger a real-time promotional offer to customers if their balances pass a certain threshold.

One issue with real-time systems is the way companies manage updates to databases in tandem with the need

to utilize a data cache. If updates occur midstream to the database from an external source, the IMDG cache providing real-time processing will suddenly be out of date. Oracle Coherence provides GoldenGate HotCache to solve these types of challenges.

Keeping data synchronized with Oracle GoldenGate HotCache

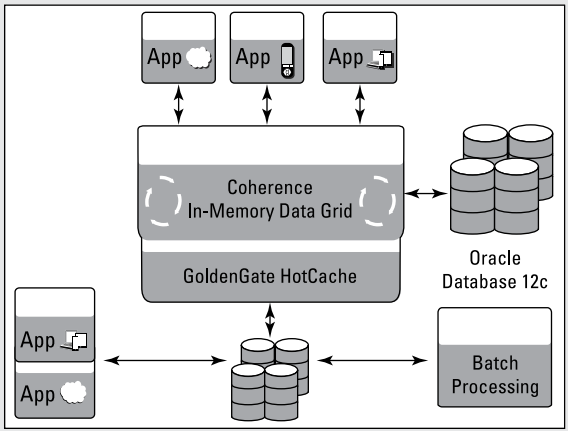
One of the challenges with using IMDGs with databases is keeping the data in the IMDG in synch as the underlying database is updated. In-memory Data Grids in general provide mechanisms for updating the database when updates are made to caches, but dealing with changes made directly to the database from other applications is more difficult to handle. Techniques such as expiration or periodic reloads of the database leave a window when the cache may have stale data, leading to outdated information delivery. Maintaining data synchronization between the IMDG and the database is complex and is difficult to implement using home-grown programmatic solutions.

Fortunately, Oracle's IMDG solution (Oracle Coherence) has tight integration with the Oracle GoldenGate solution (creating GoldenGate HotCache), allowing changes in a database to be propagated to the IMDG's cache. This action keeps the cache in synch with the underlying database at all times.

You can see in the following figure where Oracle GoldenGate HotCache is positioned.

(continued)

(continued)



As shown, Oracle GoldenGate HotCache is positioned to maintain up-to-date synchronization between Oracle Coherence IMDGs and databases. This is a major benefit to organizations trying to integrate caching solutions into existing IT infrastructure because it solves a key synchronization issue where databases are updated directly by applications. HotCache updates the data grid when a database change is made — without the need for any programming.

Chapter 3

Leveraging the Benefits of IMDGs

In This Chapter

- Gaining the most value from IMDGs
- Improving the customer experience
- Integrating IMDGs within your existing infrastructure and preferred toolsets
- Seeing how IMDGs can help a business and its customers

In-Memory Data Grids (IMDGs) bring many technical improvements to a computer system, but knowing how to incorporate IMDGs into both new and existing systems is crucial. Identifying the ways to deploy IMDGs will open many opportunities for a business if the system architect is aware of the options available. Leveraging existing infrastructure and taking advantage of IMDG integration options will allow for an easier and more effective implementation.

This chapter looks at what it takes to implement IMDGs and what benefits both the business and its customers can expect to enjoy.

Lowering Cost and Providing Better Customer Experience

Leveraging caching and memory-grid technologies has a direct impact on the customer experience. Specifically, you can expect implementation of IMDGs to improve

- **Performance:** Lessening the time the user waits for results.
- **Reliability:** Reducing the impact of technical failures on users.
- **Scalability:** Increasing processing capacity to sustain expected growth and protect against unpredicted usage spikes.

Obviously, improvements in performance, reliability, and scalability are important, but many technologies promise to do that, so why use IMDGs? The unique benefits of IMDGs relate to their application tier deployment:

- Caching occurs at the application tier where it has a greater and more noticeable impact on the user.
- Caching at the application tier with IMDGs is less expensive than boosting performance on other tiers.
- Scaling application middle-tier servers is easier operationally and far less expensive than attempting to scale enterprise-level database servers, mainframes, or partner services.

Your application may depend on one or more partner services to satisfy customer requests. For example, a travel aggregator depends on hotel, car rental, and airline partners to search for availability and pricing information on behalf

of customers. Calling partner web services makes your customer's experience dependent on external variables, such as the web services availability and responsiveness of your partner. Your partner may even charge you for using its web services. In such a case, caching results from the partner services improves customer experience and reduces costs.

IMDGs alleviate data growth, scaling, and processing bottlenecks in a system by caching the most important data and processing on the application tier. IMDGs also mask (via fault tolerance) problems that would otherwise impact the user. Via IMDGs, companies are finding the least expensive way to provide better end-user experiences to their customers.

As the amount of data increases, so must the ability to process that data increase in an effective manner.

Providing a Highly Available, Scalable, and Responsive Middleware Architecture

Building mission-critical computer systems isn't easy, but with proper planning and technology, you can be successful. Here are some ideas to consider when building critical systems:

- Gather firm requirements at the start of the project and update the plan as requirements evolve.
- Consider using technologies built to address the set of problems you anticipate rather than older technologies built for different challenges.

- Select proven technologies supported by vendors with documented, successful track records.
- Build security into the system from the start of the project, not at the end.
- Ensure that the system will be reliable by designing fault-tolerance at every level.
- Implement clustering options at the application and data tiers to enhance scalability while improving system reliability.
- Boost performance by selecting technologies that are engineered for fast response time.
- Use tools that promote caching of data at the application tier to improve performance while absorbing processing workload from the data tier.
- Plan for future growth and workload spikes by selecting technology that scales horizontally and dynamically to meet demand.

Modern systems have multiple components, and harmonious integration between components is important. IMDGs play an important part in ensuring systems that are fast, reliable, and scalable for end-users.

Integrating IMDGs with Current Infrastructure

A key driver for IMDGs is their ability to be implemented in your current system with little or no changes. Obviously, if companies had to rewrite their entire applications to use IMDGs, the adoption rates of IMDGs would be very low. First-class IMDGs can be

implemented into modern applications and support infrastructures via:

- Support for multiple programming languages to include Java, C/C++, .NET, REST (Representational State Transfer), and JSR-107 (aka JCache) interfaces.
- Ability to support myriad application tier components and application servers.
- Simplified integration with multiple database vendors and different data sources on the data tier. Some database vendors, such as Oracle, provide products to ease database integration while improving performance and synchronization.
- Plug-in availability for popular Integrated Development Environments (IDEs) to allow developers to use tools they already know.
- Management infrastructure and integration tools that simplify lifecycle management, from deployment to monitoring.

Many IMDG products are available that can benefit a business. However, when evaluating IMDG solutions, be sure to consider how easily they implement into both your existing application infrastructure and your preferred application development technology stack.

Managing Application Server HTTP Session State in IMDGs

IMDGs reside on the application tier and have the capability to manage HyperText Transmission Protocol (HTTP) session state. This allows HTTP sessions to be

shared and managed across application servers in clustered environments. Using IMDGs to manage HTTP session state provides the following benefits:

- Support more user sessions without adding more application servers
- Handle very large sessions efficiently
- Offload session and application data management from the application server tier
- Scale tiers independently
- Restart and maintain applications and web containers without losing users' session data
- Decouple session management and web containers

The capability to support a higher concurrent user base is important to companies supporting successful online businesses. The ideal solution is to leverage IMDGs with existing web and application server infrastructure to manage a larger number of concurrent users.

Many perceived performance problems are actually scalability problems. As an example, Oracle encountered a website that supported streaming events to a growing number of users. The events company found that when too many users signed up for one event, the event stream would appear jumpy. Oracle tracked this issue down to having to manage too much session state in each instance of application servers. The customer decided to use an IMDG to manage user sessions, using built-in functionality from the IMDG that didn't require any code changes in the customer application. Because application servers were no longer managing

user sessions, the servers spent less time in garbage collection and were again able to smoothly stream popular events to a large audience.

Using IMDGs in the Real World

Under what real-world circumstances would IMDGs be a good choice? The ideal use cases are situations where a larger number of distributed, concurrent users access an online system and repeatedly access similar data, in which users won't tolerate slowness or failures. Using a prior example, in this section I review an online travel and hotel reservation application used by many travelers: Prospective travelers know where they want to vacation, but the details are often in flux while they evaluate their options for the best price before committing to a purchase. Here's a list of travelers' website requirements and how IMDGs can meet them:

- Consumers, enabled by mobile devices and always-on broadband connections, access the website from around the world 24/7.
- Because the travel service is free, customers use it when they have no intent on buying. This puts strain on the website infrastructure, which increases cost to run it.
- Consumers may seek to see 100 quotes at a time, so a single query can result in aggregating a large number of quotes from a variety of sources.
- Consumers will search on similar data on competing websites as they compare flights, hotels, dates, prices, and promotions to find the deal that fits their needs the best.

- A consumer will search many times with different criteria before committing to a purchase. Consumers often compare options with other competitors. Their searches may span days or weeks.
- Performance and system stability is critical. Attempting to access data directly from each source without the benefit of caching is far too slow and prone to failures. If consumers become frustrated, they'll go to a competitor.
- Workload and usage spikes occur as online promotions are offered, special events are advertised, or seasonal events come up, even something as simple as summer vacation months.
- Caching helps insulate online travel systems from the cost, scalability, availability, and performance issues of accessing partner services.

IMDGs are a natural fit for any website that provides access to data to a growing user population, often driven by mobile or multiple-device access. Cost-effective IMDG solutions are leveraged to ensure that customers have an optimal user experience, which directly translates to increased sales and repeat business.

Oracle integration with Coherence

Oracle provides a broad spectrum of products that effectively leverage Oracle's IMDG solution, which is Coherence. Oracle has multiple products designed to easily integrate with and optimize Coherence.

Oracle supports the Coherence IMDG solution with:

- **Oracle WebLogic Server (WLS):** Coherence supports HTTP Session State management using WebLogic Server and also provides simplified lifecycle operations by using WebLogic container and management features. ***Note:*** Coherence doesn't require WebLogic Server to run.
- **Oracle Applications and Fusion Middleware Products:** Multiple Oracle products, such as PeopleSoft and JD Edwards, and other products, such as Oracle Event Processing (OEP) and Oracle Service Bus, integrate seamlessly with Coherence.
- **Oracle SOA Suite:** Oracle's Service Oriented Software suite of tools can utilize Coherence.
- **Oracle Exalogic Elastic Cloud:** Oracle's Engineered System melds hardware and software to optimize Fusion Middleware products using Coherence.
- **Oracle GoldenGate:** Synchronization between the database and Coherence cache is maintained by GoldenGate HotCache.
- **Oracle TopLink:** Provides integration capabilities with Coherence using the Java Persistence API (JPA).
- **Oracle Enterprise Manager (EM):** Administrators can manage Coherence deployments with the EM suite of management tools.
- **Oracle Database:** Oracle databases integrate with Coherence-powered applications.

If it looks like there are many Oracle tools that support IMDGs and specifically Coherence, that's because there are!

Chapter 4

Understanding Distributed Caching and In-Memory Data Grids

In This Chapter

- Describing what distributed caching is and why it is important
- Defining In-Memory Data Grids
- Explaining the topology of caching architecture
- Understanding processing and eventing within In-Memory Data Grids

Caching algorithms have long played a role in improving computer performance, and many computer systems and programs employ caching methodologies in one form or another. Within In-Memory Data Grids (IMDGs), *distributed caching* is the enabling technology. Understanding the role of caching is critical to being able to effectively use IMDGs. This chapter examines how caching is the foundation of IMDG technology and how IMDGs work from a technical perspective.

Fundamentals of Caching

People commonly place objects that they need quickly and/or use frequently near their work areas, staging them for easy access. Surgeons in operating rooms stage the tools they need before they start cutting; mechanics get the tools they need from their toolboxes before they start to work, and so on. The concept of staging what you need or will use frequently is analogous in the computing world to *caching*.

Boosting performance with caching

Because retrieving data from back-end systems and converting it is relatively slow and sometimes expensive, it's more effective to keep commonly used data in closer and much faster memory, rather than going to the back-end system each time you need to retrieve it. The time savings of keeping a copy of frequently accessed data in fast memory rather than making repetitive trips to back-end systems is measured in orders of magnitude.

Defining In-Memory Data Grids

IMDGs are defined as middleware software that manages data objects across multiple distributed servers in a horizontally scaled architecture. Specifically, an IMDG is a distributed cluster of server processes running middleware software to cache data objects to assist the application tier with faster, more reliable processing.

IMDGs provide a shared data cache across distributed servers to bring data out of remote databases and onto the application tier where it is closer to the user and closer to the application procession engines. This

ensures that the data is available on demand for the application. Caching and replication algorithms ensure that the data cached is current and copies of data are effectively managed between distributed cache servers.

Distributed caching

Although the term *caching* is broadly used and commonly understood, *distributed caching* is a somewhat newer paradigm. This type of caching indicates that a group of servers works in unison to share the load of caching all the data. To a web application, the distributed cache looks like one system, but behind the scenes, servers can be added to and removed from the system to meet the data size and throughput requirements of the application.

As an example, imagine that you're architecting an upgrade for a system that supports a large user base; for instance, a retail bank offering millions of people credit card, checking, saving, and loan services. Customers log in to your systems more and more often — driven by mobile banking apps and easy broadband access. They expect immediate responses. This extra traffic places a strain on your back-end systems. Because the amount of data is very large — in many cases several terabytes — one server can't handle all the data. This is a situation where you need a distributed cache.

When you store the frequently accessed data — current balances, recent transactions, account information — on one distributed cache, the web applications don't need to make multiple calls to various systems every time a user logs in. Instead, data is available and ready to use immediately.

Several key caching concepts are specific to distributed caching and IMDGs:

- Distributed caching and IMDGs cache data objects that would otherwise come from databases or other sources, such as other computer systems, mainframes, web services, and Big Data.
- Distributed caching and IMDGs cache data objects on the application tier for faster processing rather that storing it on the web or presentation tier.
- Unlike a cache, where you expect data to be evicted or otherwise lost, many IMDG use cases rely on never losing data.
- IMDGs support distributed computing, event-based applications, and map-reduce style aggregations on the data grid itself.

MapReduce is a programming model that uses distributed computing techniques in a cluster to process large amounts of data.

Figure 4-1 shows where IMDGs fit in your architecture.

As you can see in Figure 4-1, the IMDG exists on the application tier and is caching data objects retrieved from databases, mainframes, and web services. The IMDG is also holding state, session, and intermediate live data. Processing of the data objects occurs within the application tier, which is where the application logic occurs.

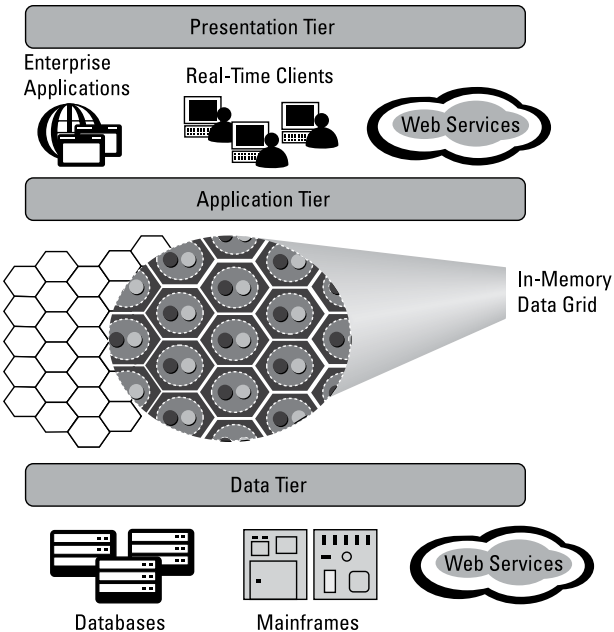


Figure 4-1: In-Memory Data Grids architecture.

Application servers are the enabling technology for the application tier. And IMDGs are often integrated with application servers. To be successful, IMDG integration with application servers should do the following:

- Support a wide range of application technologies and coding languages.
- Integrate easily within the application tier.
- Transparently support the processing on the application tier.

Progressive IMDGs are being integrated with a wide array of middleware products such as enterprise service busses, Business Process Management (BPM), and so on. Application server technology is improving, which will simplify IMDGs and operations management.

Querying Your Cache

Caching provides the greatest performance benefits by keeping objects in the application-domain format, keeping it physically closer to the application, and by scaling. Leveraging the cache within the IMDGs grid is based on several fundamental principles:

- **Getting current data locally:** IMDGs provide the flexibility to optimize data access via various cache topologies, including the capability to keep the cache local to the application process.
- **Parallel query processing:** IMDGs provide a queryable fabric capable of executing parallel queries across terabytes of data across hundreds of nodes in real time. You can spread the workload across multiple processing nodes and caches.
- **Data must be current and correct:** Unlike caches, IMDGs rely on accurate and complete data to return consistent query results. IMDGs cannot drop data or processing, even when cluster members are lost.

IMDGs are engineered to follow these principles to improve performance and provide data integrity.

Processing and Eventing

Caching is the foundation of IMDGs, but IMDGs also have two aspects that further enhance performance capabilities. These aspects are *processing* and *eventing*.

Processing

In most modern application architectures, application processing is done in the application server tier. Even with a distributed cache, the network between the application server and the distributed cache can be overwhelmed by excessive data movement.

With IMDGs, instead of bringing data objects back to the application server, you can do the processing within the memory grid itself rather than moving to and from the application server tier.

Consider, for example, an eCommerce vendor managing online shopping carts and inventory. When a user adds an item to the shopping cart, rather than moving the shopping cart, pricing, and inventory into the application to be processed, the data grid can update total shopping cart value, check inventory, and recalculate estimated shipment date and shipment costs without moving any data around.

Event-based processing

Event-based processing (*eventing* for short) allows you to execute a course of action when an object changes on the data grid. It is closely related to the type of processing discussed in the previous section, but happens

in response to different events. Continuing with the shopping cart example, when you click the purchase button, the shopping cart gets marked as purchased, and listeners cause various actions to happen: fulfilling the order and perhaps making suggestions for further purchases.

A more complex example is with electronic trading systems taking *bids* (bidding price) and *asks* (asking price) for stock trades. Every time the system gets a new bid or ask, a listener on that object within the memory grid performs a matching operation to see if the bid and ask match so that a sale can be made. If a match occurs, that match triggers further processing of the sale to occur.

Data grids are used for this type of eventing and processing because they provide inherently scalable, fault-tolerant architecture. This scalability becomes more important as the number of users and devices grows as users perform more transactions online, and as users demand faster responses and grow intolerant of outages.

Chapter 5

Ten (Okay, Eight) Things to Consider when Adopting In-Memory Data Grids

In This Chapter

- Identifying key items to consider when thinking about In-Memory Data Grids

This chapter highlights important factors that will impact the success of your In-Memory Data Grid (IMDG) and your end-users' experiences.

Ensuring That Your System Is Suitable for IMDGs

Many middleware applications will benefit from an IMDG, but some special cases exist in which the benefits won't be as pronounced. For example, an IMDG can't by itself magically reduce an inherently linear compute-intensive workload from hours down to minutes, although it could point you in the direction to make the work parallel, which would solve the problem. Before beginning an

IMDG project, examine the architecture of the existing system to see where IMDGs can be leveraged. No major undertaking should commence without a careful study, including an architecture review.

Supporting Distributed Computing

Distributed computing support is important for IMDGs deployed to support real-world systems. Deploying IMDGs across a distributed environment provides multiple benefits:

- Bringing live data closer to the applications that need it for better performance
- Reducing the amount of data movement during processing
- Providing fault tolerance in the event of a localized failure
- Spreading out the workload processing across multiple servers

When designing your IMDG architecture, be sure to account for distributed computing and leverage its benefits.

Preparing for Special Sales Events

Is it sufficient to plan only for the average daily workload and just endure the ramifications of periodic spikes? Or is it better to over-allocate processing capacity, knowing that on most days you're paying for unused capacity? For many industries, neither option is good.

A better solution is to use technology that can dynamically scale to absorb workload spikes (such as during Black Friday or the annual online shopping tradition of Cyber Monday) but shrink after the workload reduces to normal levels. Rapid provisioning of IMDGs and elastic cloud computing allow businesses to strike the right balance, allocating appropriate capacity for normal workload with being able to capitalize on surges in demand.

Making a Good First Impression

Studies show that people form lasting impressions of a company within seconds of navigating its website. Impressive User Interfaces (Uls) combined with intuitive navigation and fast response times give users assurance that they're dealing with a reputable, professional company they can trust.

IMDGs can't make a website more attractive visually or alter business processes, but they can make sure that the user has a fast, error-free experience. Preventing a company from playing damage control because of slow or error-prone applications yields many dividends to savvy companies when making their first impression on customers.

Examining Hardware Choices When Building Your Cache

Servers used to support horizontal scaling for distributed caches need to be provisioned rapidly and may be required in large numbers, but they don't need to be overly powerful in terms of individual computing capacity.

These characteristics make it possible to utilize lower cost commodity-level servers that can be procured quickly and inexpensively to join a larger grid or cluster of cache servers.

Although customers can build systems using lower-cost servers, companies like Oracle take optimizations further with engineered systems that design hardware (such as Oracle's Exalogic) and IMDG technologies (such as Coherence) to provide speed, availability, and total cost of ownership (TCO) optimizations that create a uniquely integrated platform with heightened performance, which can't be realized through use of commodity hardware.

Embracing a Single-System Image

Where the data is actually stored and processed should be non-applicable to both applications and programmers, who don't need to know or care where data and processing reside. Well-designed IMDGs make live data appear local and unified to the application, which doesn't need to be concerned with the complexity of the underlying infrastructure.

Applications accessing the data grid don't need to be aware of where data is stored across the cluster, and as partitions and objects are moved, this movement is transparent to the application. Applications access the dynamic set of live data that they need and the details are hidden in the underlying infrastructure of the IMDGs.

Identifying Live Data

Not all of your data is live data, and simply treating all data as live data is a losing proposition. For example, if you're a telecommunications company, you may have 30 million subscribers, only 5 million of whom actively use your online services. In some cases, it may be acceptable for some of your infrequent users to have to wait the first time they log in after six months (as seldom-used data is retrieved from traditional storage).

By applying an 80/20 rule to determine your critical requirements, you can realize the benefits of deploying In-Memory Data Grids more cost-effectively rather than attempting to provide instant access to infrequent and relatively lower-value users.

Understanding That Change Is Here

Right now, you may be facing scalability, offloading, and processing challenges. And, as mobile device usage, cloud computing, and the Internet of Things rapidly gain popularity, your challenges will only grow with them. Apathy about these industry trends will only make the problems worse. A different approach, one that leverages the power of IMDGs, can solve many of your problems. With IMDGs, you're in the driver's seat and not just reacting and playing catch-up to technology changes.

Notes

Use IMDGs for increased scalability and availability!

In-Memory Data Grids (IMDGs) bring data caching and processing to the application tier to make computer systems faster, more reliable, and highly scalable. Data grids allow companies to meet the evolving needs of their customers.

- ***Understand performance challenges — and why data caching is the solution***
- ***Enhance system availability and scalability — with clustering and a fault-tolerant architecture***
- ***Keep customers happy — and create new ones by giving users the experience they want***

Open the book and find:

- **How to integrate an IMDG with your existing infrastructure**
- **How to create systems that scale to meet workload requirements**
- **How to create highly available systems**
- **Why caching on the application tier improves the user experience**

Oracle engineers hardware and software to work together in the cloud and in your data center. For more information about Oracle (NYSE:ORCL), visit oracle.com.

FOR DUMMIES®
A Wiley Brand

ISBN: 978-1-118-92141-8
Not for resale