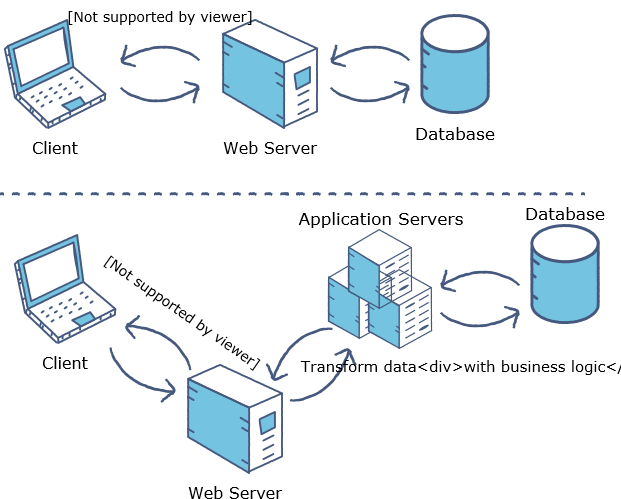
**Chapter 5. Space-Based Architecture**

Most web-based business applications follow the same general request flow: a request from a browser hits the web server, then an application server, then finally the database server.

بیشتر برنامه های کاربردی تجاری مبتنی بر وب از همان جریان درخواست عمومی پیروی می کنند: یک درخواست از یک مرورگر به سرور وب ، سپس یک سرور برنامه ، و سپس سرانجام سرور پایگاه داده ضربه می زند.



While this pattern works great for a small set of users, bottlenecks start appearing as the user load increases, first at the web-server layer, then at the application-server layer, and finally at the database-server layer.

در حالی که این الگوی برای تعداد کمی از کاربران عالی عمل می کند ، با افزایش بار کاربر ، تنگناها شروع می شوند ، ابتدا در لایه وب سرور ، سپس در لایه برنامه-سرور و در آخر در لایه بانک اطلاعات سرور ظاهر می شوند.

The usual response to bottlenecks based on an increase in user load is to scale out the web servers.

پاسخ معمول به تنگناها بر اساس افزایش بار کاربر ، مقیاس بندی سرورهای وب است.

This is relatively easy and inexpensive, and sometimes works to address the bottleneck issues.

این کار نسبتاً آسان و ارزان است و بعضی اوقات برای رفع مشکلات تنگنا کار می کند.

However, in most cases of high user load, scaling out the web-server layer just moves the bottleneck down to the application server.

با این حال ، در بیشتر موارد با بار زیاد کاربر ، مقیاس بندی لایه وب سرور فقط تنگنا را به سمت سرور برنامه حرکت می دهد.

Scaling application servers can be more complex and expensive than web servers and usually just moves the bottleneck down to the database server, which is even more difficult and expensive to scale.

مقیاس گذاری سرورهای برنامه می توانند پیچیده تر و گرانتر از سرورهای وب باشند و معمولاً فقط تنگنا را به سمت سرور پایگاه داده سوق می دهند ، که مقیاس آن حتی سخت تر و گران تر نیز هست.

Even if you can scale the database, what you eventually end up with is a triangle-shaped topology, with the widest part of the triangle being the web servers (easiest to scale) and the smallest part being the database (hardest to scale).

حتی اگر می توانید بانک اطلاعاتی را اندازه گیری کنید ، آنچه در نهایت به آن می پردازید توپولوژی مثلثی است که وسیع ترین قسمت مثلث آن ها سرورهای وب (ساده ترین به مقیاس) و کوچکترین بخش آن پایگاه داده (سخت ترین به مقیاس) است.

In any high-volume application with an extremely large concurrent user load, the database will usually be the final limiting factor in how many transactions you can process concurrently.

در هر برنامه پر حجم با باربری همزمان بسیار زیاد کاربر ، بانک اطلاعاتی معمولاً عامل محدود کننده نهایی در تعداد معاملاتی است که می توانید همزمان انجام دهید.

While various caching technologies and database scaling products help to address these issues, the fact remains that scaling out a normal application for extreme loads is a very difficult proposition.

در حالی که فن آوری های ذخیره سازی مختلف و محصولات مقیاس گذاری بانک اطلاعاتی به رفع این مشکلات کمک می کند ، واقعیت این است که مقیاس بندی یک نرم افزار معمولی برای بارهای شدید یک قضیه بسیار دشوار است.

The space-based architecture pattern is specifically designed to address and solve scalability and concurrency issues.

الگوی معماری مبتنی بر فضا به طور خاص برای رفع و حل مسائل مقیاس پذیری و همزمانی طراحی شده است.

It is also a useful architecture pattern for applications that have variable and unpredictable concurrent user volumes.

همچنین یک الگوی معماری مفید برای برنامه هایی است که دارای حجم همزمان کاربر متغیر و غیرقابل پیش بینی هستند.

Solving the extreme and variable scalability issue *architecturally* is often a better approach than trying to scale out a database or retrofit caching technologies into a non-scalable architecture.

حل مسئله مقیاس پذیری شدید و متغیر از نظر معماری ، اغلب رویکردی بهتر از تلاش برای مقیاس بندی یک بانک اطلاعاتی یا فن آوری های ذخیره سازی مقاوم سازی در یک معماری غیر مقیاس پذیر است.

**Pattern Description**

The space-based pattern (also sometimes referred to as the cloud architecture pattern) minimizes the factors that limit application scaling.

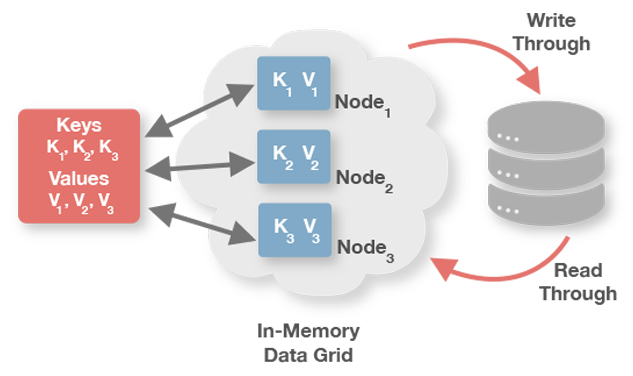
الگوی مبتنی بر فضا (که گاهی اوقات به عنوان الگوی معماری ابر نیز از آن یاد می شود) عواملی را که باعث کاهش مقیاس برنامه می شوند ، به حداقل می رساند.

This pattern gets its name from the concept of *tuple space,* the idea of distributed shared memory.

این الگوی نام خود را از مفهوم فضای تاپل گرفته است. ایده توزیع حافظه مشترک.

High scalability is achieved by removing the central database constraint and using replicated in-memory data grids instead.

مقیاس پذیری بالا با از بین بردن محدودیت پایگاه داده مرکزی و در عوض استفاده از شبکه داده های حافظه تکرار شده حاصل می شود.



Application data is kept in-memory and replicated among all the active processing units.

داده های برنامه در حافظه نگه داشته شده و در بین واحدهای پردازش فعال همانند سازی می شوند.

Processing units can be dynamically started up and shut down as user load increases and decreases, thereby addressing variable scalability.

با افزایش و کاهش بار کاربر ، واحدهای پردازشی بصورت پویا راه اندازی و تعطیل می شوند و در نتیجه به مقیاس پذیری متغیر می پردازند.

Because there is no central database, the database bottleneck is removed, providing near-infinite scalability within the application.

از آنجا که هیچ پایگاه داده مرکزی وجود ندارد ، تنگنا پایگاه داده برداشته می شود ، مقیاس پذیری تقریباً نامتناهی را در داخل برنامه فراهم می کند.

Most applications that fit into this pattern are standard websites that receive a request from a browser and perform some sort of action.

اکثر برنامه های کاربردی که در این الگوی قرار می گیرند ، وب سایتهای استاندارد هستند که درخواستی را از یک مرورگر دریافت می کنند و به نوعی عمل می کنند.

A bidding auction site is a good example of this. یک سایت حراج مناقصه نمونه خوبی از این مورد است.

The site continually receives bids from internet users through a browser request. The application would receive a bid for a particular item, record that bid with a timestamp, and update the latest bid information for the item, and send the information back to the browser.

این سایت دائماً از طریق درخواست مرورگر ، پیشنهاداتی را از کاربران اینترنت دریافت می کند. برنامه برای یک مورد خاص یک پیشنهاد دریافت می کند ، آن پیشنهاد را با یک نشانگر زمانی ضبط می کند ، و آخرین اطلاعات پیشنهاد را برای این مورد به روز می کند ، و اطلاعات را به مرورگر ارسال می کند.

There are two primary components within this architecture pattern: a *processing unit* and *virtualized middleware*. [Figure 5-1](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch05.html#sapr_0501_img) illustrates the basic space-based architecture pattern and its primary architecture components.

دو المان اصلی در این الگوی معماری وجود دارد: یک واحد پردازش و واسط مجازی. شکل 5-1 الگوی اساسی معماری مبتنی بر فضا و اجزای اصلی معماری آن را نشان می دهد.

The processing-unit component contains the application components (or portions of the application components).

بخش واحد پردازش شامل اجزای برنامه (یا بخش هایی از اجزای برنامه) است.

This includes web-based components as well as backend business logic.

این شامل مؤلفه های مبتنی بر وب و همچنین منطق تجارت با پس زمینه است

The contents of the processing unit varies based on the type of application—smaller web-based applications would likely be deployed into a single processing unit, whereas larger applications may split the application functionality into multiple processing units based on the functional areas of the application.

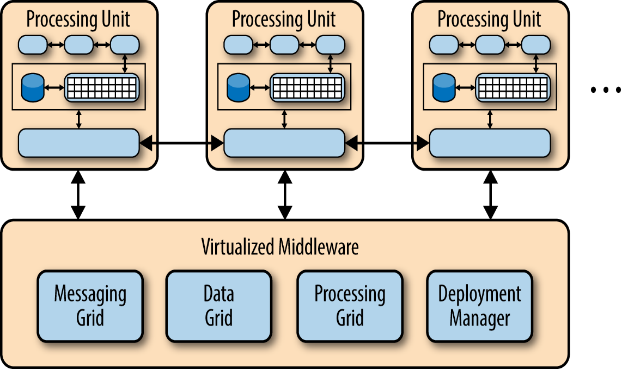
محتویات واحد پردازش بسته به نوع برنامه متفاوت است — برنامه های کوچکتر مبتنی بر وب احتمالاً در یک واحد پردازش واحد مستقر می شوند ، در حالی که برنامه های بزرگتر ممکن است عملکرد برنامه را بر اساس مناطق کاربردی برنامه به چندین واحد پردازش تقسیم کنند.

The processing unit typically contains the application modules, along with an in-memory data grid and an optional asynchronous persistent store for failover.

واحد پردازش به طور معمول شامل ماژول های برنامه ، به همراه یک شبکه داده حافظه و یک انبار ذخیره اختیاری ناهمزمان برای عدم موفقیت است.

It also contains a replication engine that is used by the virtualized middleware to replicate data changes made by one processing unit to other active processing units.

همچنین حاوی یک موتور همانندسازی است که توسط middleware مجازی مورد استفاده قرار می گیرد تا داده های ایجاد شده توسط یک واحد پردازش به سایر واحدهای پردازش فعال را تکرار کند.



**Figure 5-1. Space-based architecture pattern**

The virtualized-middleware component handles housekeeping and communications.

## (housekeeping - Computer Definition

The instructions that are executed at the beginning of an application. It resets all counters and flags to their starting values and readies the program for user operation.

دستورالعمل هایی که در ابتدای برنامه اجرا می شوند. همه پیشخوان ها و پرچم ها را به مقادیر شروع شده خود تنظیم می کند و برنامه را برای عملکرد کاربر آماده می کند.

)

It contains components that control various aspects of data synchronization and request handling.

این شامل مؤلفه هایی است که جنبه های مختلف هماهنگ سازی داده ها و رسیدگی به درخواست را کنترل می کند.

Included in the virtualized middleware are the messaging grid, data grid, processing grid, and deployment manager. These components, which are described in detail in the next section, can be custom written or purchased as third-party products.

در میان افزار مجازی شامل شبکه پیام رسانی ، شبکه داده ، شبکه پردازش و مدیر استقرار هستند. این مؤلفه ها که در بخش بعدی به تفصیل شرح داده شده است ، می توانند به صورت سفارشی به عنوان محصولات شخص ثالث نوشته یا خریداری شوند.

**Pattern Dynamics**

The magic of the space-based architecture pattern lies in the virtualized middleware components and the in-memory data grid contained within each processing unit.

جادوی الگوی معماری مبتنی بر فضا در اجزای میان افزار مجازی و شبکه داده های حافظه موجود در هر واحد پردازش قرار دارد.

[Figure 5-2](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch05.html#sapr_0502_img) shows the typical processing unit architecture containing the application modules, in-memory data grid, optional asynchronous persistence store for failover, and the data-replication engine.

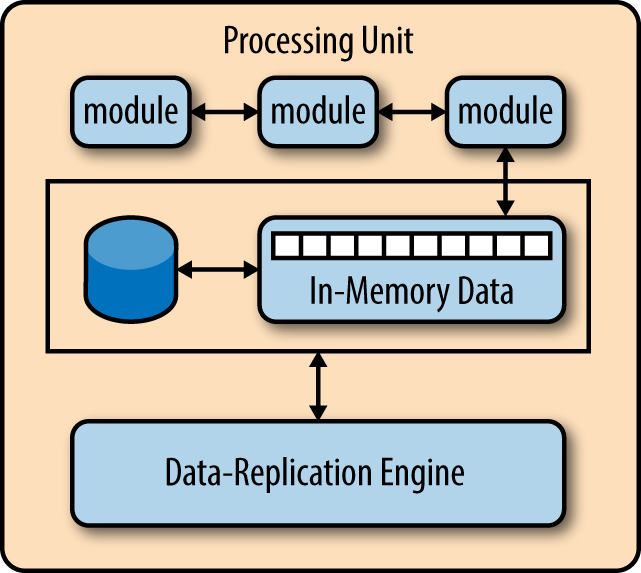
شکل 5-2 معماری واحد پردازش معمولی حاوی ماژول های برنامه ، شبکه داده های حافظه ، فروشگاه پایدار ناهمزمان اختیاری برای عدم موفقیت ، و موتور تکرار داده را نشان می دهد.

The virtualized middleware is essentially the controller for the architecture and manages requests, sessions, data replication, distributed request processing, and process-unit deployment.

میان افزار مجازی سازی شده در اصل کنترلر معماری است و درخواست ها ، جلسات ، تکثیر داده ها ، پردازش درخواست تقسیم شده و استقرار واحد پردازش را مدیریت می کند.

There are four main architecture components in the virtualized middleware: the messaging grid, the data grid, the processing grid, and the deployment manager.

چهار عنصر اصلی معماری در میان افزار مجازی وجود دارد: شبکه پیام رسانی ، شبکه داده ، شبکه پردازش و مدیر استقرار.



**Figure 5-2. Processing-unit component**

**Messaging Grid**

The messaging grid, shown in [Figure 5-3](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch05.html#sapr_0503_img), manages input request and session information. When a request comes into the virtualized-middleware component, the messaging-grid component determines which active processing components are available to receive the request and forwards the request to one of those processing units.

شبکه پیام رسانی ، که در شکل 5-3 نشان داده شده است ، درخواست درخواست ورودی و اطلاعات جلسه را مدیریت می کند. هنگامی که یک درخواست به مؤلفه میان افزار مجازی شده مجازی وارد می شود ، مؤلفه شبکه پیام رسانی تعیین می کند کدام قسمت پردازش فعال برای دریافت درخواست در دسترس است و درخواست را به یکی از آن واحد پردازش ارسال می کند.

The complexity of the messaging grid can range from a simple round-robin algorithm to a more complex next-available algorithm that keeps track of which request is being processed by which processing unit.

پیچیدگی شبکه پیام رسانی می تواند از یک الگوریتم round-robin تا یک الگوریتم پیچیده next-available باشد که پیگیری درخواست را توسط کدام واحد پردازش، پردازش می کند.

**Data Grid**

The data-grid component is perhaps the most important and crucial component in this pattern.

مؤلفه شبکه داده شاید مهمترین و مهمترین مؤلفه در این الگو باشد.

The data grid interacts with the data-replication engine in each processing unit to manage the data replication between processing units when data updates occur.

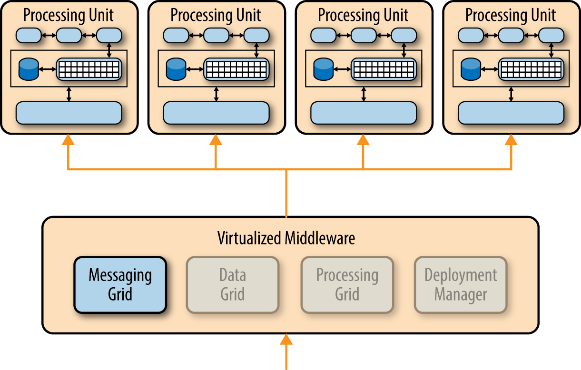
شبکه داده با موتور تکثیر داده در هر واحد پردازش در تعامل است تا بتواند هنگام بروزرسانی داده ها داده را بین واحدهای پردازش مدیریت کند.

Since the messaging grid can forward a request to any of the processing units available, it is essential that each processing unit contains *exactly* the same data in its in-memory data grid.

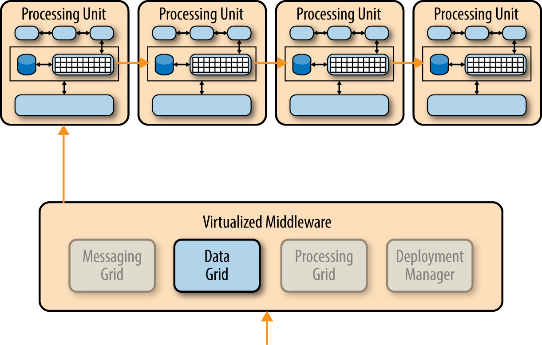
از آنجا که شبکه پیام رسانی می تواند درخواست را به هر یک از واحدهای پردازش موجود ارسال کند ، ضروری است که هر واحد پردازش دقیقاً همان داده را در شبکه داده حافظه خود داشته باشد.

Although [Figure 5-4](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch05.html#sapr_0504_img) shows a synchronous data replication between processing units, in reality this is done in parallel asynchronously and *very quickly*, sometimes completing the data synchronization in a matter of microseconds (one millionth of a second).

اگرچه در شکل 5-4 تکثیر داده همزمان در بین واحدهای پردازش نشان داده شده است ، اما در واقعیت این کار به صورت موازی و همزمان و خیلی سریع انجام می شود ، گاهی اوقات هماهنگ سازی داده ها در یک ماده میکرو ثانیه (یک میلیونم ثانیه) انجام می شود.



**Figure 5-3. Messaging-grid component**

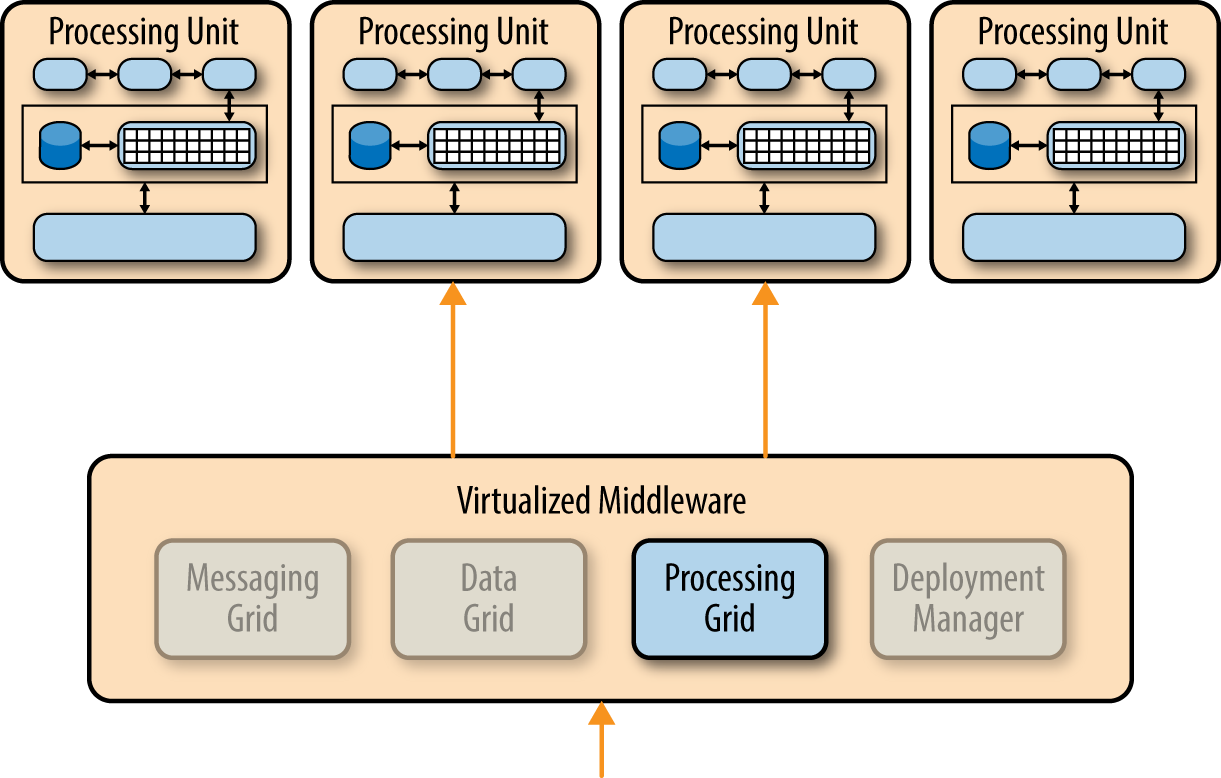


**Figure 5-4. Data-grid component**

**Processing Grid**

The processing grid, illustrated in [Figure 5-5](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch05.html#sapr_0505_img), is an optional component within the virtualized middleware that manages distributed request processing when there are multiple processing units, each handling a portion of the application. If a request comes in that requires coordination between processing unit types (e.g., an order processing unit and a customer processing unit), it is the processing grid that mediates and orchestrates the request between those two processing units.

شبکه پردازش ، که در شکل 5-5 نشان داده شده است ، یک مؤلفه اختیاری در میان افزار مجازی است که پردازش درخواست توزیع شده را هنگامی که چندین واحد پردازش وجود دارد ، مدیریت می کند ، هر کدام بخشی از برنامه را کنترل می کنند. اگر درخواستی وارد شود که نیاز به هماهنگی بین انواع واحد پردازش (به عنوان مثال ، واحد پردازش سفارش و یک واحد پردازش مشتری) داشته باشد ، این شبکه پردازشی است که درخواست را بین این دو واحد پردازش میانجی و هماهنگ می کند.



**Figure 5-5. Processing-grid component**

**Deployment Manager**

The deployment-manager component manages the dynamic startup and shutdown of processing units based on load conditions. This component continually monitors response times and user loads, and starts up new processing units when load increases, and shuts down processing units when the load decreases. It is a critical component to achieving variable scalability needs within an application.

مؤلفه استقرار مدیر عامل راه اندازی و خاموش کردن واحدهای پردازشی پویا را بر اساس شرایط بار مدیریت می کند. این مؤلفه بطور مداوم زمان پاسخ و بارهای کاربر را کنترل می کند و با افزایش بار واحد های پردازش جدید را راه اندازی می کند و با کاهش بار واحد پردازش را خاموش می کند. این یک جزء مهم برای دستیابی به نیازهای مقیاس پذیری متغیر در یک برنامه است.

**Considerations**

The space-based architecture pattern is a complex and expensive pattern to implement. It is a good architecture choice for smaller web-based applications with variable load (e.g., social media sites, bidding and auction sites). However, it is not well suited for traditional large-scale relational database applications with large amounts of operational data.

الگوی معماری مبتنی بر فضا الگویی پیچیده و گران برای پیاده سازی است. این یک معماری مناسب برای برنامه های کاربردی مبتنی بر وب کوچکتر با بار متغیر است (به عنوان مثال ، سایت های رسانه های اجتماعی ، سایت های مناقصه و حراج). با این حال ، آن را مناسب برای برنامه های کاربردی پایگاه داده بزرگ در مقیاس بزرگ با مقادیر زیادی از داده های عملیاتی مناسب نیست.

Although the space-based architecture pattern does not require a centralized datastore, one is commonly included to perform the initial in-memory data grid load and asynchronously persist data updates made by the processing units. It is also a common practice to create separate partitions that isolate volatile and widely used transactional data from non-active data, in order to reduce the memory footprint of the in-memory data grid within each processing unit.

اگرچه الگوی معماری مبتنی بر فضا نیازی به یک مرکز داده متمرکز ندارد ، معمولاً یکی از آنها برای انجام بار اولیه شبکه داده های حافظه استفاده می شود و به طور همزمان به روز رسانی داده های ساخته شده توسط واحد پردازش را ادامه می دهد. همچنین ساختن پارتیشنهای جداگانه که دادههای معامله ناپایدار و پرکاربرد را از دادههای غیرفعال جدا می کنند ، به عنوان یک روش معمول است تا بتوانید ردپای حافظه شبکه داده های حافظه را در هر واحد پردازش کاهش دهید.

It is important to note that while the alternative name of this pattern is the cloud-based architecture, the processing units (as well as the virtualized middleware) do not have to reside on cloud-based hosted services or PaaS (platform as a service). It can just as easily reside on local servers, which is one of the reasons I prefer the name “space-based architecture.”

توجه به این نکته حائز اهمیت است که در حالی که نام جایگزین این الگوی معماری ابری مبتنی بر ابر است ، واحدهای پردازش (و همچنین واسطه مجازی) مجبور نیستند روی سرویس های میزبانی شده مبتنی بر ابر یا PaaS (سیستم عامل به عنوان سرویس) اقامت داشته باشند. . این به راحتی می تواند بر روی سرورهای محلی اقامت داشته باشد ، که یکی از دلایلی است که من نام "معماری مبتنی بر فضا" را ترجیح می دهم.

From a product implementation perspective, you can implement many of the architecture components in this pattern through third-party products such as GemFire, JavaSpaces, GigaSpaces, IBM Object Grid, nCache, and Oracle Coherence. Because the implementation of this pattern varies greatly in terms of cost and capabilities (particularly data replication times), as an architect, you should first establish what your specific goals and needs are before making any product selections.

از منظر اجرای محصول ، می توانید بسیاری از مؤلفه های معماری را در این الگوی از طریق محصولات شخص ثالث مانند GemFire ، JavaSpaces ، GigaSpaces ، IBM Object Grid ، nCache و Oracle Coherence پیاده سازی کنید. از آنجا که اجرای این الگوی از نظر هزینه و قابلیت ها (به خصوص زمان تکثیر داده ها) بسیار متفاوت است ، به عنوان یک معمار ، ابتدا باید قبل از انتخاب هر محصول ، اهداف و نیازهای خاص خود را مشخص کنید.

**Pattern Analysis**

The following table contains a rating and analysis of the common architecture characteristics for the space-based architecture pattern. The rating for each characteristic is based on the natural tendency for that characteristic as a capability based on a typical implementation of the pattern, as well as what the pattern is generally known for. For a side-by-side comparison of how this pattern relates to other patterns in this report, please refer to [Appendix A](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/app01.html#pattern-analysis-summary) at the end of this report.

جدول زیر حاوی رتبه بندی و تجزیه و تحلیل ویژگی های معماری معمول برای الگوی معماری مبتنی بر فضا است. امتیاز برای هر مشخصه بر اساس گرایش طبیعی به آن ویژگی به عنوان یک قابلیت مبتنی بر اجرای معمولی از الگوی ، و همچنین آنچه که این الگوی به طور کلی برای آن شناخته شده است ، بستگی دارد. برای مقایسه جانبی ، چگونگی ارتباط این الگوی با سایر الگوهای موجود در این گزارش ، لطفاً در پایان این گزارش به پیوست A مراجعه کنید.

**Overall agility**

*Rating:* High

*Analysis:* Overall agility is the ability to respond quickly to a constantly changing environment. Because processing units (deployed instances of the application) can be brought up and down quickly, applications respond well to changes related to an increase or decrease in user load (environment changes).  Architectures created using this pattern generally respond well to coding changes due to the small application size and dynamic nature of the pattern.

چابکی کلی توانایی پاسخ دادن سریع به یک محیط مداوم در حال تغییر است. از آنجا که واحدهای پردازشی (موارد مستقر در برنامه) می توانند به سرعت بالا و پایین کشیده شوند ، برنامه ها به تغییرات مربوط به افزایش یا کاهش بار کاربر (تغییرات محیط) پاسخ خوبی می دهند. معماری های ایجاد شده با استفاده از این الگوی به دلیل کوچک بودن کاربرد و ماهیت پویای این الگوی ، به طور کلی به تغییرات کدگذاری پاسخ می دهند.

**Ease of deployment**

*Rating:* High

*Analysis:* Although space-based architectures are generally not decoupled and distributed, they are dynamic, and sophisticated cloud-based tools allow for applications to easily be “pushed” out to servers, simplifying deployment.

اگرچه معماریهای مبتنی بر فضا به طور کلی از بین نرفته و توزیع نمی شوند ، اما پویا هستند و ابزارهای پیشرفته مبتنی بر ابر این امکان را فراهم می آورند که برنامه ها را به راحتی به سرورها سوق دهید و استقرار را ساده کنید.

**Testability**

*Rating:* Low

*Analysis:* Achieving very high user loads in a test environment is both expensive and time consuming, making it difficult to test the scalability aspects of the application.

دستیابی به بارهای بسیار زیاد کاربر در یک محیط تست ، هم پرهزینه است و هم وقت گیر است ، و آزمایش جنبه های مقیاس پذیری برنامه را دشوار می کند.

**Performance**

*Rating:* High

*Analysis:* High performance is achieved through the in-memory data access and caching mechanisms build into this pattern.

عملکرد بالا از طریق دسترسی به داده های حافظه و مکانیسم های حافظه پنهانی که در این الگوی ایجاد می شوند ، حاصل می شود.

**Scalability**

*Rating:* High

*Analysis:* High scalability come from the fact that there is little or no dependency on a centralized database, therefore essentially removing this limiting bottleneck from the scalability equation.

مقیاس پذیری بالا از این واقعیت ناشی می شود که وابستگی کمی یا هیچ وابستگی به یک بانک اطلاعاتی متمرکز وجود ندارد ، بنابراین اساساً این تنگنای محدودکننده از معادله مقیاس پذیری حذف می شود.

**Ease of development**

*Rating:* Low

*Analysis:* Sophisticated caching and in-memory data grid products make this pattern relatively complex to develop, mostly because of the lack of familiarity with the tools and products used to create this type of architecture. Furthermore, special care must be taken while developing these types of architectures to make sure nothing in the source code impacts performance and scalability.

محصولات شبکه ذخیره سازی پیشرفته و شبکه های حافظه ، این الگوی را نسبتاً پیچیده تر توسعه می دهند ، بیشتر به دلیل عدم آشنایی با ابزارها و محصولات مورد استفاده در ایجاد این نوع معماری. علاوه بر این ، باید در هنگام تهیه این نوع معماری ها دقت خاصی شود تا اطمینان حاصل شود که هیچ چیزی در کد منبع عملکرد و مقیاس پذیری را تحت تأثیر قرار نمی دهد.

<https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch05.html>