Architecture of Enterprise Applications V Quality Attributes

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

Why do we should consider QA?

- Business considerations determine qualities that must be accommodated in a system's architecture.
  - These qualities are over and above that of functionality, which is the basic statement of the system's capabilities, services, and behavior
- Systems are frequently redesigned not because they are functionally deficient, but because they are difficult to maintain, port, or scale, or are too slow, or have been compromised by network hackers.
  - the replacements are often functionally identical

Functionality and Architecture

- Functionality and quality attributes are orthogonal
  - this is not to say that any level of any quality attribute is achievable with any function.
  - any of functions your choices as an architect will determine the relative level of quality

- What is functionality?
  - It is the ability of the system to do the work for which it was intended.

- Functionality may be achieved through the use of any of a number of possible structures.

Architecture and Quality Attributes

- Achieving quality attributes must be considered throughout design, implementation, and deployment.
  - No quality attribute is entirely dependent on design, nor is it entirely dependent on implementation or deployment.
- Satisfactory results are a matter of getting the big picture (architecture) as well as the details (implementation) correct. For example:
  - Usability
  - Modifiability
  - Performance

Architecture and Quality Attributes

- The message of this section is twofold:
  - Architecture is critical to the realization of many qualities of interest in a system, and these qualities should be designed in and can be evaluated at the architectural level.
  - Architecture, by itself, is unable to achieve qualities. It provides the foundation for achieving quality, but this foundation will be to no avail if attention is not paid to the details.

Architecture and Quality Attributes

- Within complex systems, quality attributes can never be achieved in isolation. The achievement of any one will have an effect, sometimes positive and sometimes negative, on the achievement of others.
  - security and reliability
  - almost every quality attribute negatively affects performance

- We will examine the following three classes:
  - Qualities of the system. We will focus on availability, modifiability, performance, security, testability, and usability.
  - Business qualities (such as time to market) that are affected by the architecture.
  - Architecture qualities, such as conceptual integrity, that are about the architecture itself although they indirectly affect other qualities, such as modifiability.

Quality Attribute Scenarios

- A quality attribute scenario is a quality-attribute-specific requirement. It consists of six parts.
  - Source of stimulus. This is some entity (a human, a computer system, or any other actuator) that generated the stimulus.
  - Stimulus. The stimulus is a condition that needs to be considered when it arrives at a system.
  - Environment. The stimulus occurs within certain conditions. The system may be in an overload condition or may be running when the stimulus occurs, or some other condition may be true.
  - Artifact. Some artifact is stimulated. This may be the whole system or some pieces of it.
  - Response. The response is the activity undertaken after the arrival of the stimulus.
  - Response measure. When the response occurs, it should be measurable in some fashion so that the requirement can be tested.

Architecture of Enterprise Applications V Availability

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

AVAILABILITY

- Availability is concerned with system failure and its associated consequences.
  - system failure occurs when the system no longer delivers a service consistent with its specification. Such a failure is observable by the system's users—either humans or other systems.

- Among the areas of concern are
  - how system failure is detected
  - how frequently system failure may occur
  - what happens when a failure occurs
  - how long a system is allowed to be out of operation
  - when failures may occur safely
  - how failures can be prevented
  - what kinds of notifications are required when a failure occurs

AVAILABILITY

- We need to differentiate between failures and faults.
  - A fault may become a failure if not corrected or masked. That is, a failure is observable by the system's user and a fault is not. When a fault does become observable, it becomes a failure.
  - For example, a fault can be choosing the wrong algorithm for a computation, resulting in a miscalculation that causes the system to fail.

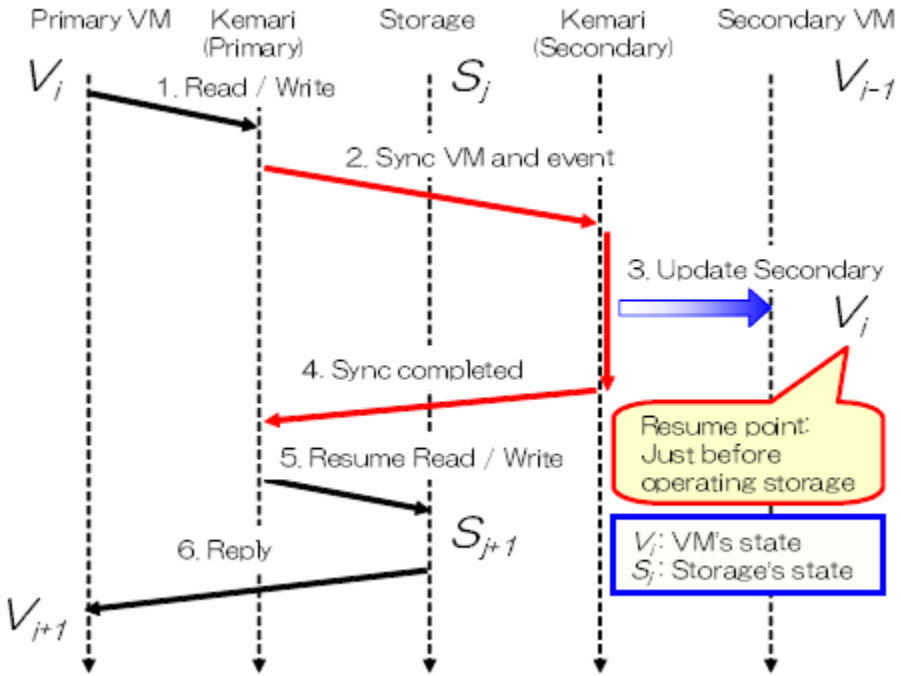
- Once a system fails, an important related concept becomes the time it takes to repair it.
  - Since a system failure is observable by users, the time to repair is the time until the failure is no longer observable.

AVAILABILITY

- The distinction between faults and failures allows discussion of automatic repair strategies.
  - That is, if code containing a fault is executed but the system is able to recover from the fault without it being observable, there is no failure.

AVAILABILITY

- For example：
  - Kemari
  - http://www.osrg.net/kemari/
  - Kemari: Virtual Machine Synchronization for Fault Tolerance
  - http://wiki.xensource.com/xenwiki/Open_Topics_For_Discussion?action=AttachFile&do=get&target=Kemari_08.pdf



AVAILABILITY

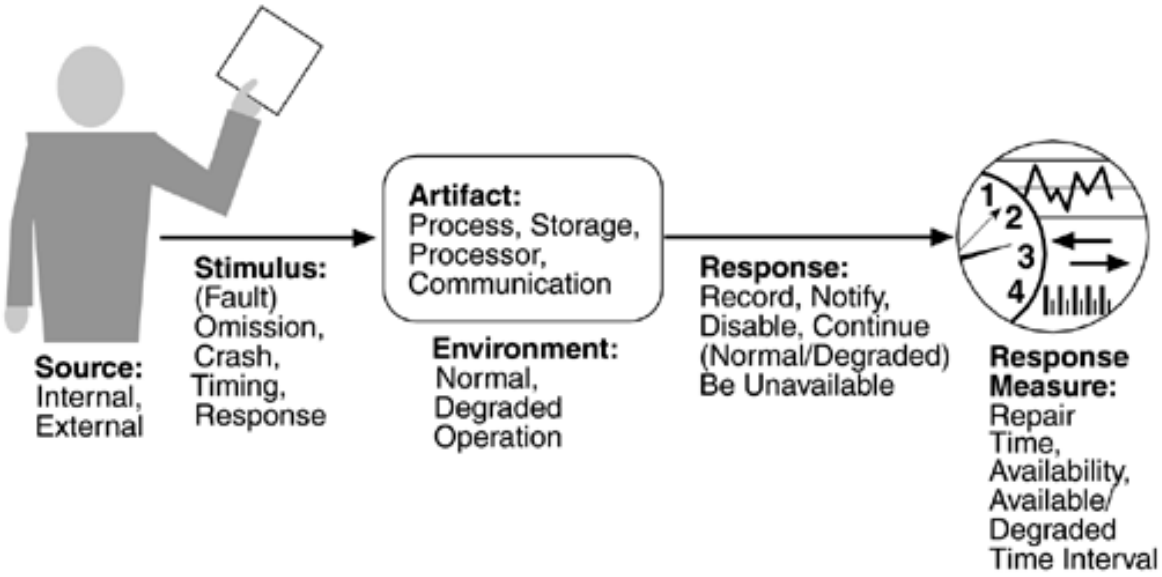
- Source of stimulus.
  - We differentiate between internal and external indications of faults or failure since the desired system response may be different.
- Stimulus. A fault of one of the following classes occurs.
  - omission. A component fails to respond to an input.
  - crash. The component repeatedly suffers omission faults.
  - timing. A component responds but the response is early or late.
  - response. A component responds with an incorrect value.
- Artifact.
  - This specifies the resource that is required to be highly available, such as a processor, communication channel, process, or storage.
- Environment.
  - The state of the system when the fault or failure occurs may also affect the desired system response. For example, if the system has already seen some faults and is operating in other than normal mode, it may be desirable to shut it down totally. However, if this is the first fault observed, some degradation of response time or function may be preferred.

AVAILABILITY

- Response.
  - There are a number of possible reactions to a system failure. These include
    - logging the failure
    - notifying selected users or other systems
    - switching to a degraded mode with either less capacity or less function
    - shutting down external systems
    - becoming unavailable during repair.

- Response measure.
  - The response measure can specify an availability percentage, or it can specify a time to repair, times during which the system must be available, or the duration for which the system must be available.

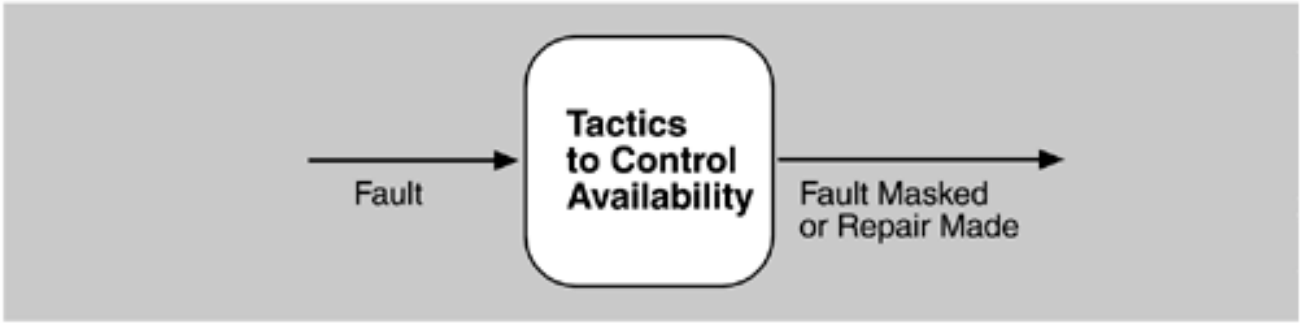
AVAILABILITY



Availability Tactics

- A failure occurs when the system no longer delivers a service that is consistent with its specification; this failure is observable by the system's users.
- A fault (or combination of faults) has the potential to cause a failure.
- Recovery or repair is an important aspect of availability.
- The tactics we discuss in this section will keep faults from becoming failures or at least bound the effects of the fault and make repair possible.

Goal of availability tactics



Availability Tactics

- Many of the tactics we discuss are available within standard execution environments such as operating systems, application servers, and database management systems.
- It is still important to understand the tactics used so that the effects of using a particular one can be considered during design and evaluation.
- All approaches to maintaining availability involve some type of redundancy, some type of health monitoring to detect a failure, and some type of recovery when a failure is detected.
  - In some cases, the monitoring or recovery is automatic and in others it is manual.
- We first consider fault detection. We then consider fault recovery and finally, briefly, fault prevention.

Availability Tactics-fault detection

- Three widely used tactics for recognizing faults are ping/echo, heartbeat, and exceptions.
  - Ping/echo.
    - One component issues a ping and expects to receive back an echo, within a predefined time, from the component under scrutiny.
    - This can be used within a group of components mutually responsible for one task.
    - It can also used be used by clients to ensure that a server object and the communication path to the server are operating within the expected performance bounds.
    - "Ping/echo" fault detectors can be organized in a hierarchy, in which a lowest-level detector pings the software processes with which it shares a processor, and the higher-level fault detectors ping lower-level ones.
    - This uses less communications bandwidth than a remote fault detector that pings all processes.

Availability Tactics-fault detection

- Heartbeat (dead man timer).
  - In this case one component emits a heartbeat message periodically and another component listens for it.
  - If the heartbeat fails, the originating component is assumed to have failed and a fault correction component is notified.
  - The heartbeat can also carry data.
- Exceptions.
  - One method for recognizing faults is to encounter an exception, which is raised when one of the fault classes is recognized.
  - The exception handler typically executes in the same process that introduced the exception.

- The ping/echo and heartbeat tactics operate among distinct processes, and the exception tactic operates within a single process.
- The exception handler will usually perform a semantic transformation of the fault into a form that can be processed.

Availability Tactics-fault detection

- 假设我们在RUBiS中希望增加对数据库管理系统的错误探测功能，以探测数据库服务器的连接错误。
  - 考虑到数据库服务器本身在系统中的压力会比较大，因此错误探测应该尽量不影响系统性能；
  - 同时，由于数据库服务器连接错误又是一个严重错误，因此我们希望提高错误探测的实时性。

- 综合考虑，我们选择使用心跳策略来实现这项错误探测功能
  - 即在系统中增加一个服务，它周期性地创建到数据库的连接，并且将连接结果以心跳方式发送给其他部件。

Availability Tactics-fault detection

- 无线电高度表-嵌入式系统
- 自检
  - 加电自检
  - 周期性自检
  - 手动自检
  - 受令自检

  - LED显示错误代码

Availability Tactics-fault recovery

- Fault recovery consists of preparing for recovery and making the system repair. Some preparation and repair tactics follow.

- Active redundancy (hot restart).
  - All redundant components respond to events in parallel. Consequently, they are all in the same state.
  - The response from only one component is used (usually the first to respond), and the rest are discarded.
  - When a fault occurs, the downtime of systems using this tactic is usually milliseconds since the backup is current and the only time to recover is the switching time.
  - Active redundancy is often used in a client/server configuration, such as database management systems, where quick responses are necessary even when a fault occurs. In a highly available distributed system, the redundancy may be in the communication paths. For example, it may be desirable to use a LAN with a number of parallel paths and place each redundant component in a separate path. In this case, a single bridge or path failure will not make all of the system's components unavailable.

Availability Tactics-fault recovery

- Passive redundancy
  (warm restart/dual redundancy/triple redundancy).
  - One component (the primary) responds to events and informs the other components (the standbys) of state updates they must make. When a fault occurs, the system must first ensure that the backup state is sufficiently fresh before resuming services.
  - This approach is also used in control systems, often when the inputs come over communication channels or from sensors and have to be switched from the primary to the backup on failure.
  - Some database systems force a switch with storage of every new data item. The new data item is stored in a shadow page and the old page becomes a backup for recovery. In this case, the downtime can usually be limited to seconds.
  - Synchronization is the responsibility of the primary component, which may use atomic broadcasts to the secondaries to guarantee synchronization.

Availability Tactics-fault recovery

- Spare.
  - A standby spare computing platform is configured to replace many different failed components.
  - It must be rebooted to the appropriate software configuration and have its state initialized when a failure occurs.
  - Making a checkpoint of the system state to a persistent device periodically and logging all state changes to a persistent device allows for the spare to be set to the appropriate state.
  - This is often used as the standby client workstation, where the user can move when a failure occurs.
  - The downtime for this tactic is usually minutes.

Availability Tactics-fault recovery

- 假设我们在RUBiS中希望增加对数据库的备份，以便在主服务器产生错误之后，能够切换到备份服务器上。
  - 考虑到RUBiS并非关键系统，因此对其可用性要求不是很高，我们允许适量的用户会话状态丢失；
  - 同时，我们要求一旦用户出价成功，那么该数据一定要妥善保存，即持久化的数据不能丢失。

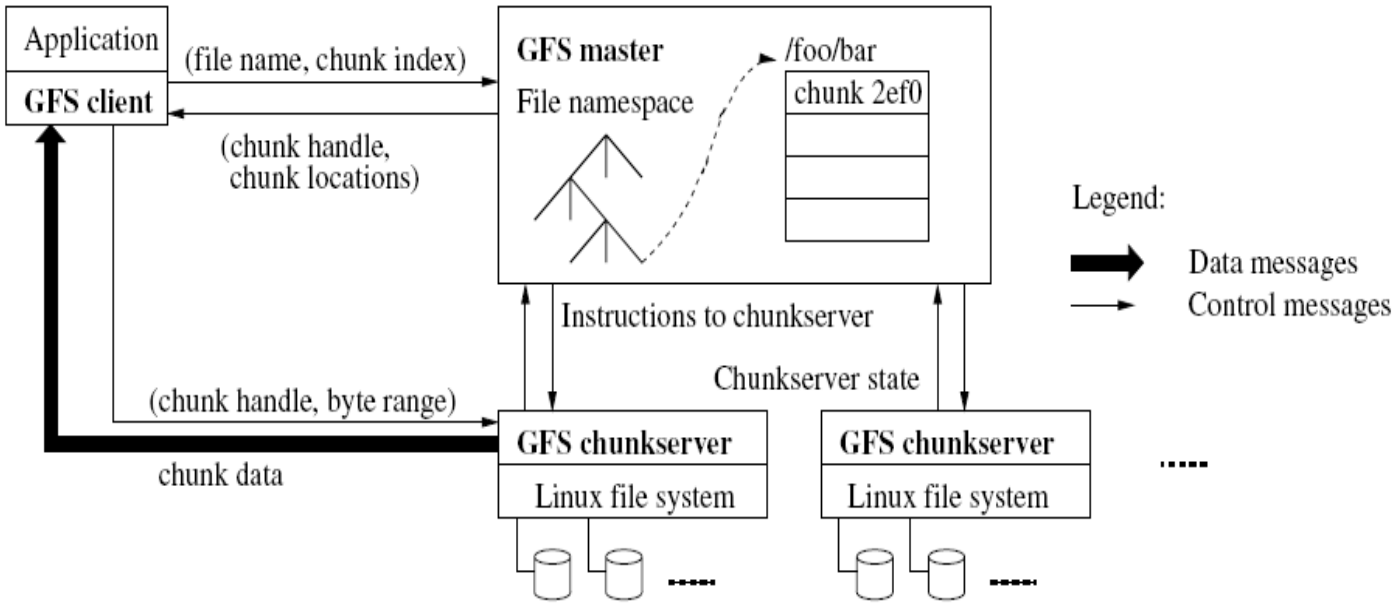
- 综合考虑，我们使用暖备份机制对数据库服务器进行备份。

Availability Tactics-fault recovery

- 无线电高度表-嵌入式系统
- 冗余
  - 存储单元冗余
  - 程序代码冗余
  - 通信接口冗余

Availability Tactics-fault recovery

- GFS
  - 每个文件分块可以有若干个备份



Availability Tactics-fault repair

- There are tactics for repair that rely on component reintroduction. When a redundant component fails, it may be reintroduced after it has been corrected..

- Shadow operation.
  - A previously failed component may be run in "shadow mode" for a short time to make sure that it mimics the behavior of the working components before restoring it to service.

- Checkpoint/rollback.
  - A checkpoint is a recording of a consistent state created either periodically or in response to specific events.
  - Sometimes a system fails in an unusual manner, with a detectably inconsistent state. In this case, the system should be restored using a previous checkpoint of a consistent state and a log of the transactions that occurred since the snapshot was taken.

Availability Tactics-fault repair

- 对于RUBiS系统，考虑到系统性能，我们采用周期性记录系统检查点的方式。
  - 对于替换下来的出错的数据库服务器，我们将使用最近的检查点对其进行修复，然后使用系统日志将此检查点之后产生的持久性操作同步到待修复的服务器中。
  - 在执行这些操作之后，被修复的数据库服务器就可以再次引入系统中了。

Availability Tactics-fault repair

- 无线电高度表-嵌入式系统
- 系统修复
  - 看门狗
  - 软中断
  - 部件热插拔

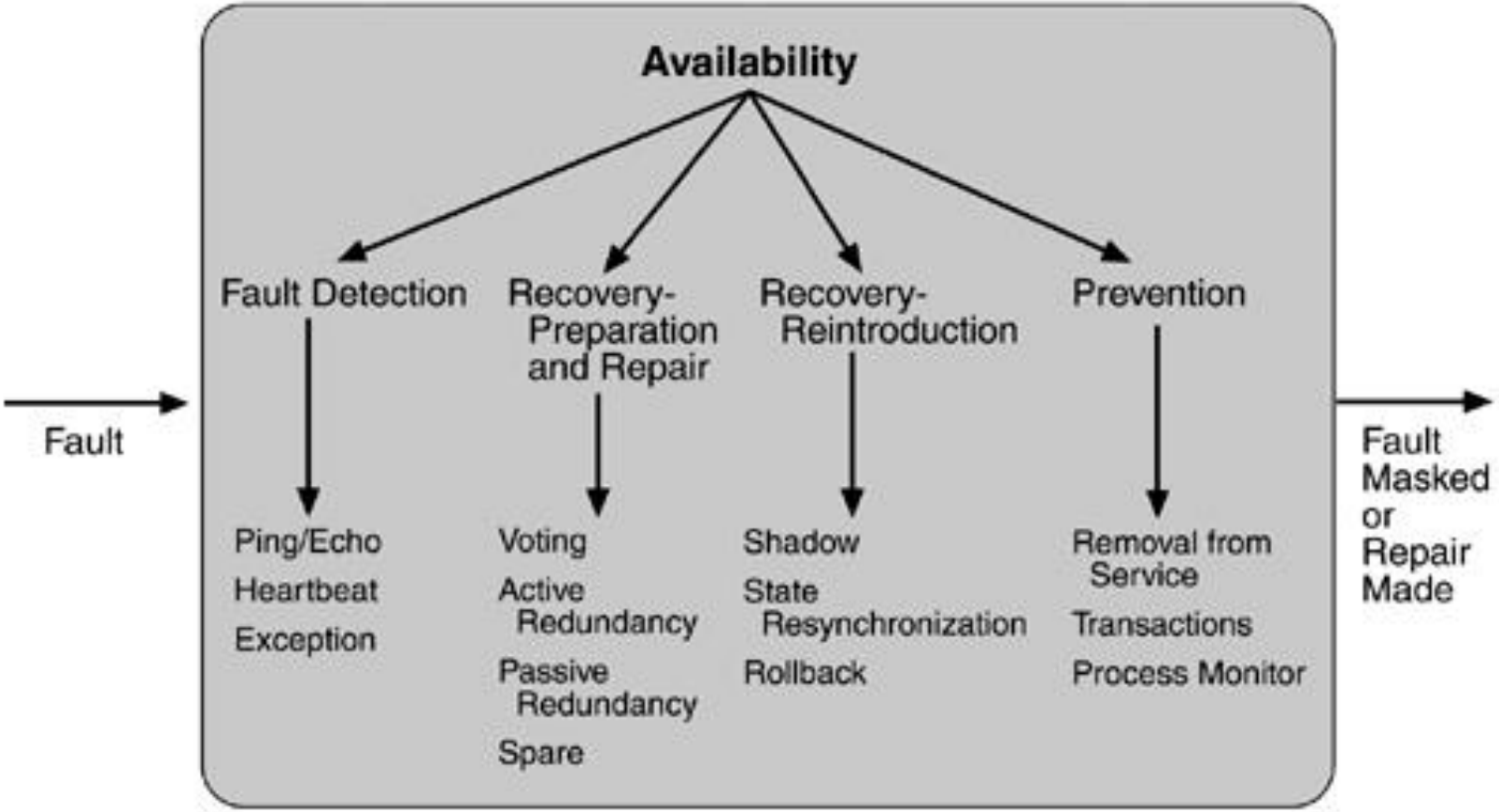
Availability Tactics-fault prevention

- The following are some fault prevention tactics.
- Removal from service.
  - This tactic removes a component of the system from operation to undergo some activities to prevent anticipated failures.
  - One example is rebooting a component to prevent memory leaks from causing a failure.
  - If this removal from service is automatic, an architectural strategy can be designed to support it. If it is manual, the system must be designed to support it.
- Transactions.
  - A transaction is the bundling of several sequential steps such that the entire bundle can be undone at once.
  - Transactions are used to prevent any data from being affected if one step in a process fails and also to prevent collisions among several simultaneous threads accessing the same data.
- Process monitor.
  - Once a fault in a process has been detected, a monitoring process can delete the nonperforming process and create a new instance of it, initialized to some appropriate state as in the spare tactic.

Availability Tactics-fault prevention

- 在RUBiS中，当出于业务考虑：
  - 我们将某个region从regions表里面删除时，由于users表引用了regions表，因此，我们希望在删除这个region时，将与其关联的所有users表的region域置为缺省值，例如null。
  - 显然，删除region记录和置空users表的region域两个操作应该捆绑执行，因此，我们将这两个动作定义为一个事务，保证其要么都执行，要么都不执行。

Availability Tactics-Summary



Reference

- Software.Architecture.In.Practice.2nd.Edition

Thank You!