TABLE OF CONTENTS

METHODS AND APPARATUSES FOR IMPROVING FAILURE RECOVERY IN A DISTRIBUTED SYSTEM

BACKGROUND

In multi-tenant distributed processing systems, it is extremely difficult to discover failures and recover from them. These systems typically log errors by capturing a stack trace and recording it to a file or data source. However, because most distributed systems lack task coordination, recovery is even trickier. In fact, without such coordination, when a failure does occur such systems cannot recover from the point at which the failure occurred in the workflow. Instead, failure recovery typically necessitates retrying the processing chain, which can be expensive depending on how much processing power is required.

SUMMARY

Example embodiments described herein address the above deficiencies and provide methods and apparatuses that enable distributed processing systems to reliably recover from failures. To accomplish this goal, example embodiments implement task coordination within the distributed system by binding various units of work together with a state machine. While the units of work do not directly interact with one another, use of a state machine to bind the units of work together ensures that the processing chain itself is stateful and enables the processing chain to effectively recover from failures.

BRIEF DESCRIPTION OF THE DRAWINGS

[0001] Having described certain example embodiments of the present disclosure in general terms above, reference will now be made to the accompanying drawings, which are not necessarily drawn to scale.

[0002] FIGS. 1A-1C illustrate a series of diagrams illustrating failure recovery in traditional distributed systems.

[0003] FIG. 2A illustrates a schematic diagram representing an example processing chain implementation utilizing a state machine, in accordance with some example embodiments described herein.

[0004] FIG. 2B illustrates a more detailed schematic block diagram of an example processing chain implementation that depicts how a distributed system may utilize a resolver to recover from a failure, in accordance with some example embodiments described herein.

[0005] FIG. 3A illustrates a high-level system diagram for implementing some example embodiments described herein.

[0006] FIG. 3B illustrates a partial view of the high-level system diagram of FIG. 3A.

[0007] FIG. 3C illustrates another partial view of the high-level system diagram of FIG. 3A.

[0008] FIG. 4 illustrates a schematic block diagram of example circuitry embodying a networked device that may perform operations in accordance with some example embodiments described herein.

[0009] FIGS 5A and 5B illustrate a particular example processing chain associated with receipt of three orders for tests to be performed at a particular lab, in accordance with some example embodiments described herein.

[0010] FIG. 6 illustrates an order state machine, in accordance with an example use case described in connection with FIGS. 5A and 5B.

[0011] FIGS. 7A and 7B illustrate a processing chain associated with failure to complete a particular test O1, in a variant of the example embodiment described in connection with FIGS 5A and 5B.

[0012] FIG. 8 illustrates an order state machine including an error state, in accordance with an example use case described in connection with FIGS. 7A and 7B.

[0013] FIG. 9 depicts how a distributed system may utilize a resolver to recover from the failure of test 01 in the example described in connection with FIGS. 7A and 7B.

[0014] FIGS. 10A and 10B illustrate a processing chain associated with recovery from failure of test O1, in a variant of the example embodiment described in connection with FIGS 5A and 5B.

[0015] FIG. 11 illustrates a flowchart describing example operations performed for improving failure recovery in a distributed system during execution of a processing chain, in accordance with some example embodiments described herein.

[0016] FIG. 12 illustrates a flowchart describing example operations for reporting errors occurring during execution of a processing chain, in accordance with some example embodiments described herein.

DETAILED DESCRIPTION

1. In FIG. 1A, a processing chain is shown for execution by a distributed system. The processing chain has three units of work (A, B, and C) that need to be executed in sequence.

2. FIG. 1B illustrates that when there is a failure while trying to to transition from unit B to unit C, the contact points between units B and C in FIG. 1B is severed. The system modelled in FIG. 1A does not have a way to transition between units of work when a failure severs the processing chain. To advance from unit B to unit C in this system thus requires the expenditure of engineering resources to deduce the following information: (1) what processing occurs in unit C to determine the proper way to reposition the system such that unit C can run properly; and (2) what data is necessary coming out of unit C to put the system in its proper state thereafter.

3. Figure 1C represents what recovery looks like in distributed systems that do not employ the concept of task coordination.

FIG. 1A

FIG. 1B

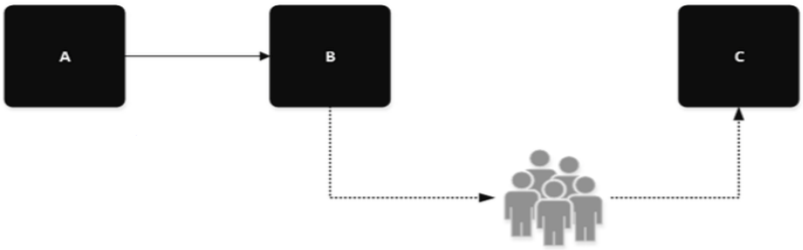


FIG. 1C

FIGS. 1A-1C illustrate a series of diagrams illustrating failure recovery in traditional distributed systems.

4. FIG. 2A illustrates a model for implementing failure recovery, in accordance with some example embodiments described herein. As shown in FIG. 2A, this model adds a state machine 202 that binds together the units of work A, B, and C in the processing chain. Each unit of work in the processing chain is represented by a corresponding state (in this example, one of states S1, S2, S3, or Error), a state transition (in this example, one of transitions T1, T2, T2', or T3), and an event (in this example, one of events E1, E2, E3, E5, or E6). The individual units of work A, B, and C shown in FIG. 2A do not themselves store the context describing how to work with each other. Instead, the state machine is responsible for documenting this context. Keeping track of the state of the system in this fashion therefore establishes a fundamental capability of the model illustrated in FIG. 2A that is not present in the model shown in FIGS. 1A-1C: the model shown in FIG. 2A is stateful.

5. In FIG. 2A, when an error occurs at event E5 (as with the examples shown in FIGS. 1A-1C, FIG. 2A shows an error occurring between units of work B and C), the state machine 202 of FIG. 2A will transition to the error state, which serves as a logical stopping point whereby processing can continue later during the recovery phase. When this happens, a failure recovery system will record what is called an “incident.”

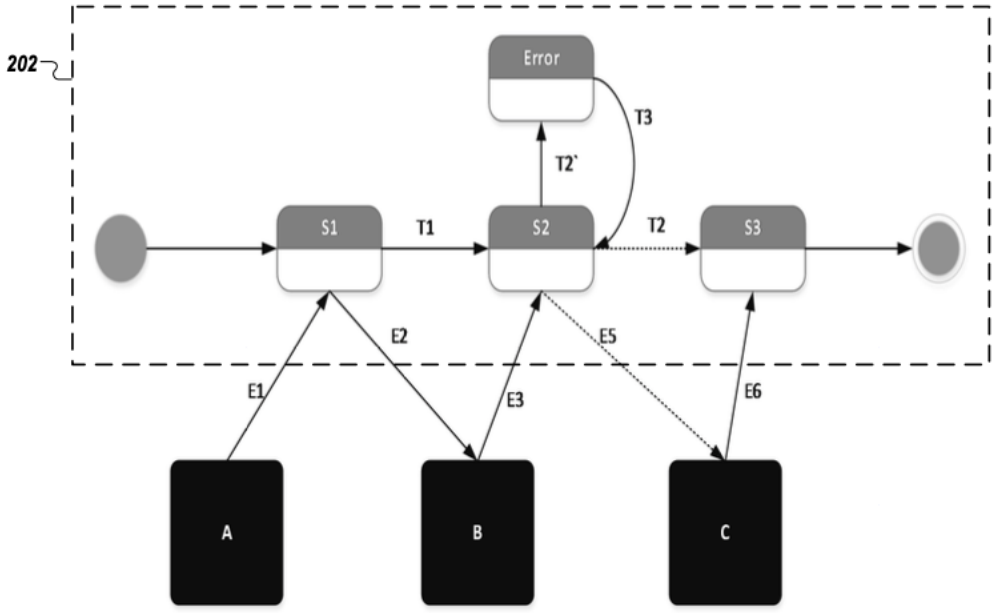


FIG. 2A

6. In this regard, attention will be turned now FIG. 2B, which provides a schematic block diagram showing how a distributed system can utilize a resolver 204 to facilitate recovery from a failure in accordance with some example embodiments described herein. Applying the resolver concept (which is discussed in greater detail in accordance with FIGS. 3A-3C) to the model shown in FIG. 2A, the model shown in FIG. 2B illustrates the delegation of the job of instructing the system on how to recover from the error to resolver 204, which is shown in FIG. 2B as part of a failure recovery system that may be distinct from the system implementing the processing chain and corresponding state machine. It will be noted that it remains the responsibility of humans to define what points in the system are recoverable and how to recover from failure at those points. However, after the resolver 204 has been registered with the failure recovery system, humans are no longer needed in the recovery process for that failure scenario, except in the optional situation where it is desirable for a user to manually initiate the recovery operation.

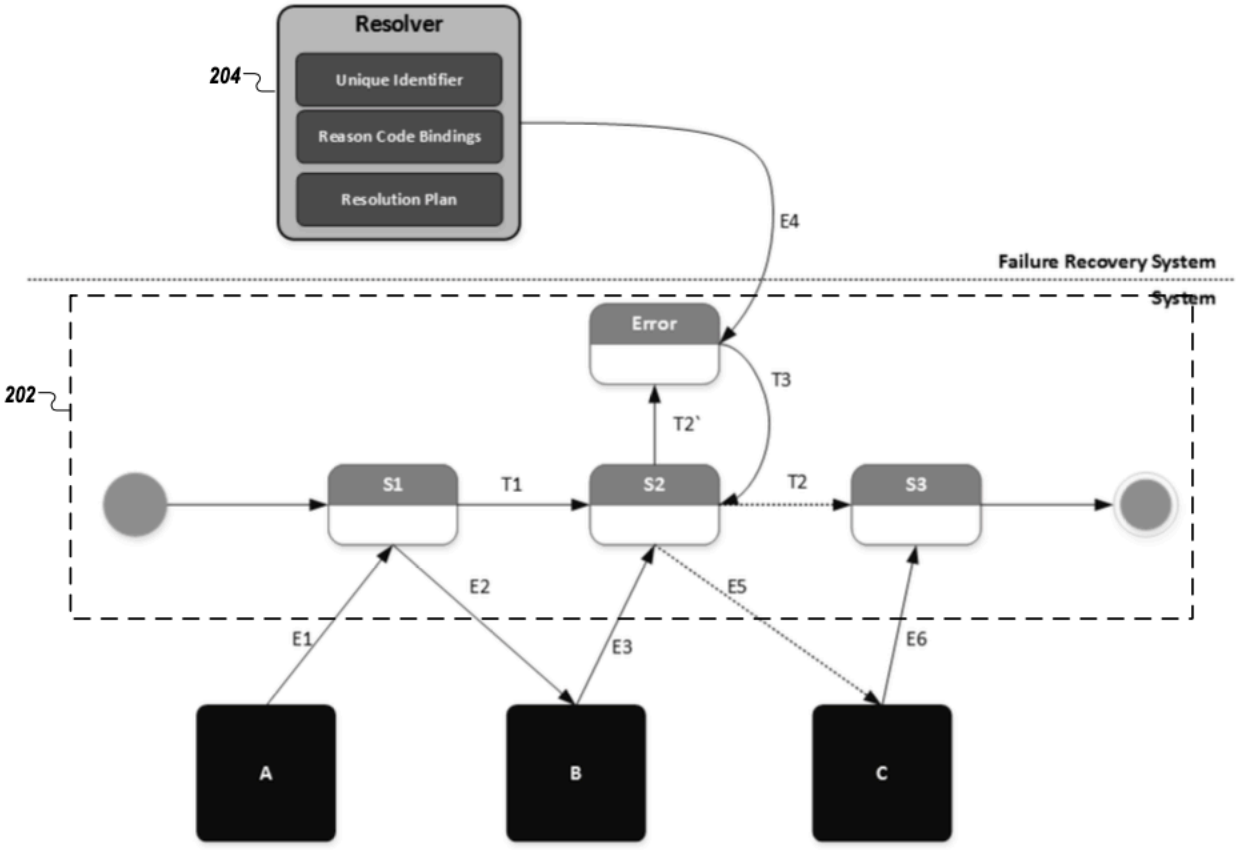


FIG. 2B

7. First, as shown by operation 308, the system may put itself into a failed state, aggregate all subsequent notifications, and then send a request to a failure report generator. In this regard, it should be understood that a "notification," as referenced here, may contain data indicating a reason code (e.g., a numerical number and description representing the context of the notification), a time stamp defining when the notification was created, a type defining which entity initiated the notification (e.g., the system or a user), a severity defining whether the notification is an error, a warning, or simply information in nature, and a criticality, which defines the level of intensity of the notification (e.g., low, medium, high, critical, or the like). In turn, the failure report generator can identify notifications that self-identify as errors and generate a corresponding incident. The corresponding incident may in turn be correlated to the source by way of a correlation ID and a reason code.

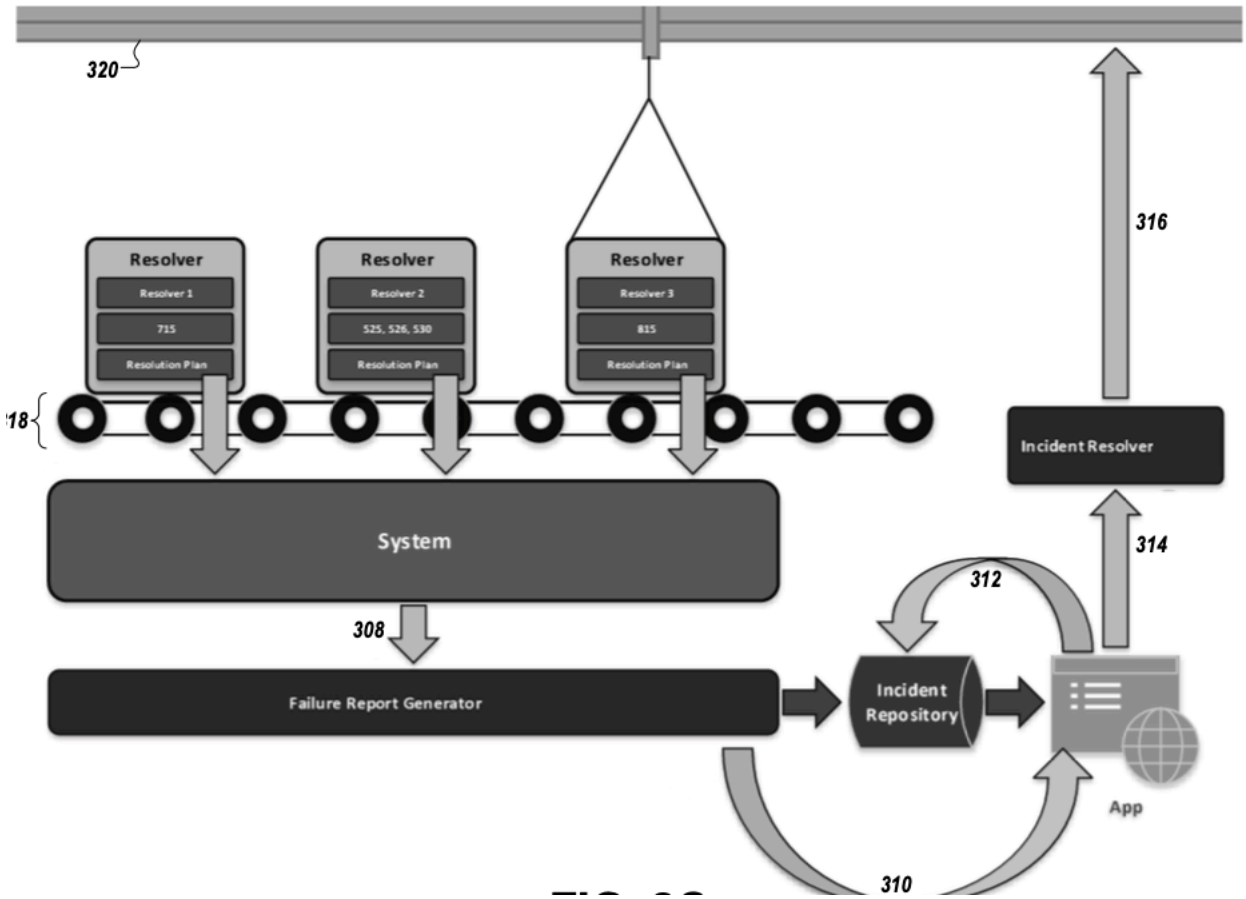
8. At operation 310, the failure report generator may thereafter notify applications that are interested in the correlation ID, which in turn will be able to recover from the failed state in the system. Once notified, at operation 312 each application will retrieve all the associated incidents given a particular correlation ID from an incident repository and may in some embodiments display them to the user for resolution. When providing such a display, the presentation may include a reason code, a summary of the issue, and detailed information on how to resolve the issue (e.g., a knowledgebase article).

9. As shown at operation 314, when a "Recovery" button is clicked by the user, the application will send a request to an incident resolver with information pertinent to build a resolution plan such as the list of incidents, the person who is initiating recovery, and the correlation ID. The user might in turn need to go to another application to make the necessary changes for successful recovery.

10. As shown at operation 316, the incident resolver may then call a resolution plan selector 320, passing it the list of incidents, for selection of the corresponding resolver for each reason code. Because a resolution plan can be bound to multiple reason codes – the caveat being that a single reason code cannot belong to multiple resolution plans – and because an incident has a one-to-one relationship with a reason code, the resolution plan selector thereafter is able to determine whether a resolver has already been selected in the scenario when the system has recorded several incidents during processing. So, plans that have already been selected are marked according to the responsive resolver's unique identifier, thereby ensuring that they will not be executed more than once.

11. As shown at operation 318, a resolution plan executor cycles through the list of resolvers, calling their corresponding resolution plan in a sequential chain without regard for order, with the caveat being that the application must be smart enough to know where to record incidents and how to respond to recovery requests. Each resolver then addresses its corresponding failures, and the system should continue processing as if it never failed from this point forward.

12. Accordingly, by implementing task coordination with a state machine, example embodiments disclosed herein create a stateful implementation of a processing chain that allows the retrieval of the context of the processing chain at a failure point, which in turn enables the processing chain to recover from failures more effectively by rehydrating relevant data fields, rather than retrying entire units of work. Furthermore, by utilizing automatic resolvers to direct this rehydration, reliance on human effort can be diminished, thus improving consistency, and reducing the potential for human error. Example methods and apparatuses that implement these concepts are described in greater detail below.


320
316
Resolver
Resolver 1
715
Resolution Plan
Resolver
Resolver 2
525, 526, 530
Resolution Plan
Resolver
Resolver 3
815
Resolution Plan
318
Incident Resolver
System
314
312
308
Failure Report Generator
Incident Repository
App
310

13. As shown in FIG. 5A, an order (ORM) may be placed with the system requesting performance of three tests (O1, O2, and O3) by way of a single ORM message (for instance, it may be the case that the settings for the lab at which the order is to be fulfilled may mandate that only a maximum of 3 orders per ORM message can be sent to that lab). As shown by operation 502, the system may identify that three tests have been ordered, and then, in operations 504A-504C and 506A-506C, the processing chain may process each of the three tests separately. Subsequently, the separate results may be accumulated in operation 508, and as shown in FIG. 5B, these results are bundled together in operation 510, and an ORM message to the consumer is created at 512, a routing slip is appended in operation 514, and the ORM message is delivered back to the consumer in operation 516. Accordingly, it should be evident from this processing flow that a single ORM message can have one or more associated tests.

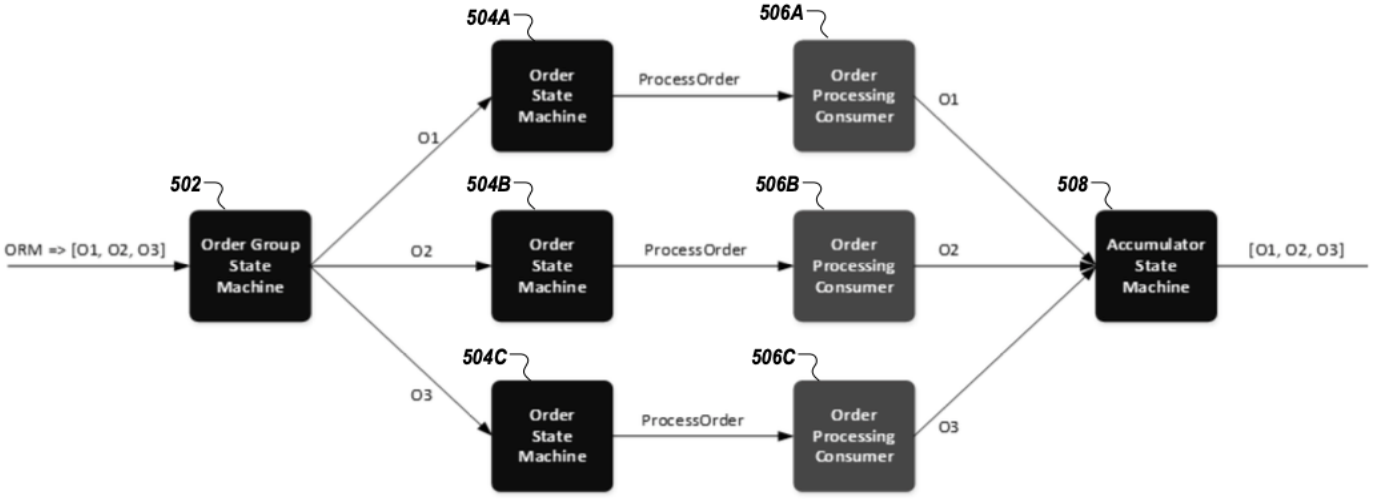


FIG. 5A

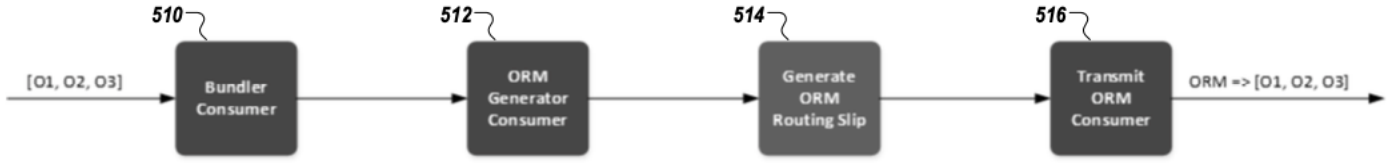


FIG. 5B

14. A corresponding order state machine may be represented by the model shown in FIG. 6.

FIG. 6

15. While FIGS. 5A and 5B illustrate an example where processing does not fail, consider the situation that occurs when test O1 fails while executing in its corresponding Orders State Machine. Assume, for now, that this failure occurred because the system could not find the ordering provider (which may, for instance, comprise reason code 645) and because there was no mapping to test O1 in the lab's catalog (which may comprise reason code 657). As shown in FIGS. 7A and 7B, in this situation test O1 would never make it to the Accumulator State Machine for processing (as shown by empty box 702), and instead test O1 will be put into an error state in its corresponding Order State Machine (as shown in FIG. 8). In the meantime, tests O2 and O3 would be processed, and in this example a corresponding ORM message would be sent to the consumer regarding those two tests alone.

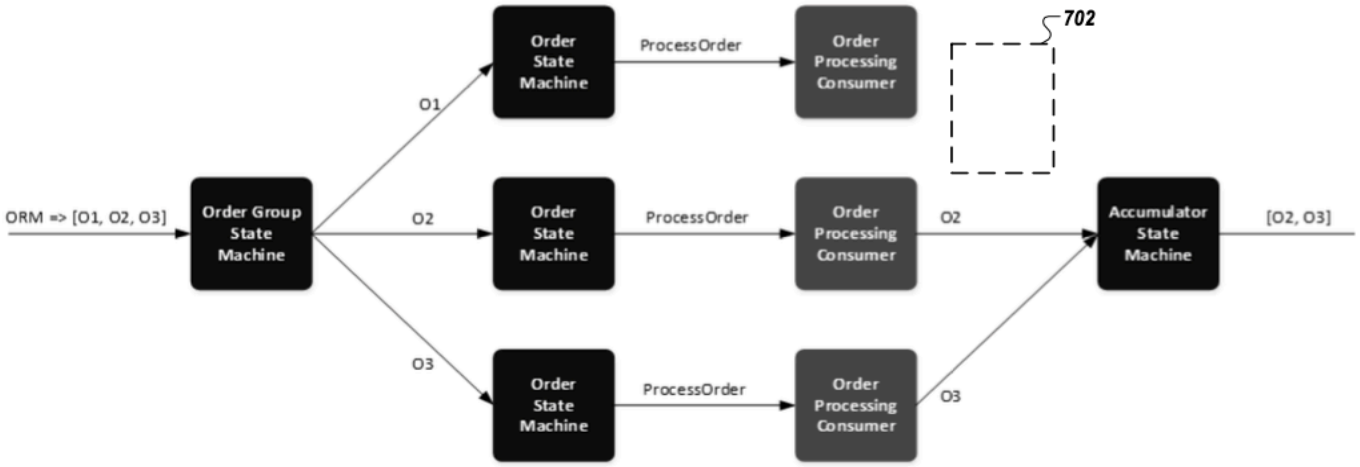


FIG. 7A

FIG. 7B

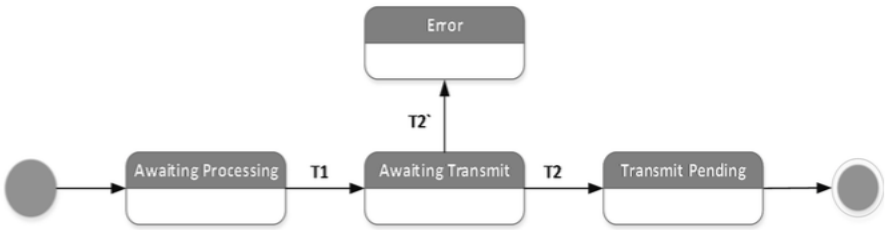


FIG. 8

16. In relation to the failure of test O1, the user would be able to see both reason codes 645 and 657 in a user interface (UI) as both error codes would both be recorded at different points in the processing pipeline yet correlated to a single identifier. In this example, time is not a factor here, and O1 will sit in the error state until recovery is initiated. When recovery is initiated (as shown in FIG. 9), an appropriate resolver would then be selected. When the resolver's resolution plan is executed, it would send a command (e.g., Orderable Rectified) to the system (Orders State Machine) that is in a state of error, which would then have enough information to ultimately rehydrate the command (i.e., Process Order) that is needed to process test O1. This event is later observed by the Order Processing Consumer as if it was initiated as part of normal processing. Since both reason codes 645 and 657 are in this example bound to the same resolver, the resolver would only be selected once. Otherwise, the system would send out an additional ORM.

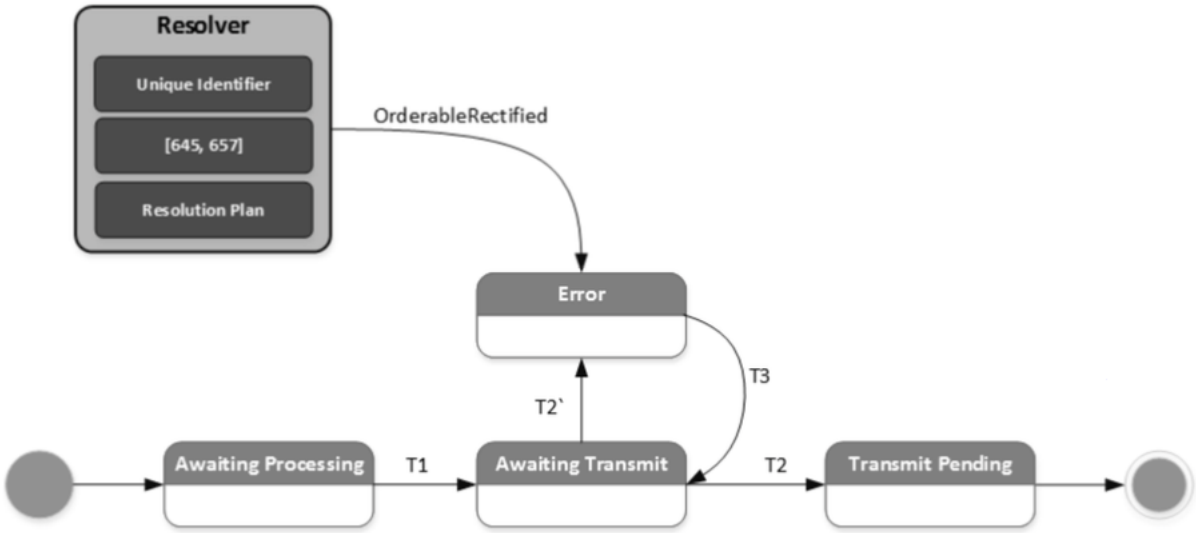


FIG. 9

17. So, once the resolution command (i.e., Orderable Rectified) has been sent, the state machine will observe this command and return to its normal state of Awaiting Transmit as it would have if no error ever happened. In this regard, resolution would have the effect of sending test O1 out on a new ORM (as shown in FIGS. 10A and 10B) since the Accumulator would not have held back the other ordered tests just because test O1 was in a failed state.

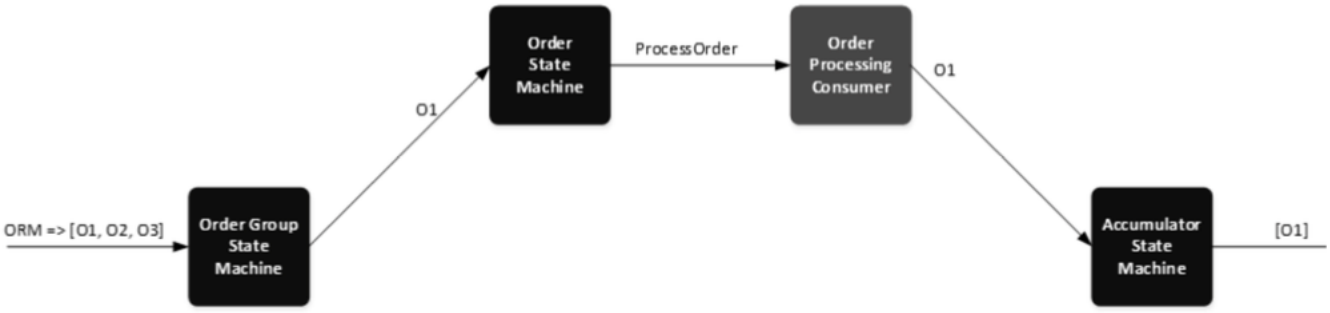


FIG. 10A

FIG. 10B

ABSTRACT OF THE DISCLOSURE

Embodiments are disclosed for improving failure recovery in a distributed system during execution of a processing chain. In the context of a method, an example embodiment includes implementing, by the distributed system, a task coordination state machine corresponding to the processing chain. This example embodiment of the method further includes discovering, using the task coordination state machine, an error occurring during execution of the processing chain. Finally, the example embodiment of the method further includes reporting the error by a failure report generator and recovering, by an incident resolver, from the error. Corresponding apparatuses and computer program products are also provided.