1. **Cơ chế quản lý giao tác đồng thời.**

Đối với Aerospike, cơ chế quản lý giao tác đồng thời được gọi là Strong Consistency. Strong Consistency đảm bảo rằng tất cả dữ liệu được ghi vào trên mỗi một dòng dữ liệu được thực thi theo một trình tự nhất định (theo thứ tự tuần tự), và các dòng dữ liệu ghi vào sẽ không bị sắp xếp lại thứ tự hoặc là bị bỏ qua..v.v các dữ liệu này sẽ không mất

Ở mọi dạng của Strong Consistency, dữ liệu ghi vào sẽ không bị mất – trong giới hạn của các lỗi phần cứng đồng thời.

Aerospike đảm bảo rằng dữ liệu sẽ không bị mất, với ba trường hợp ngoại lệ:

1. Tiến trình xử lý các node dừng sau khi quá 27 giây, hoặc thời gian chênh lệch giữa các cụm node lớn hơn 27s

2. Server đồng thời không sạch hoặc không đáng tin cậy sập nếu **commit-to-device** (cơ chế chờ cho dữ liệu ghi vào đĩa hoặc RAM) không cho phép

3. Nhiều lỗi lưu trữ phần cứng trước khi dữ liệu có thể được sao chép

Trong mỗi trường hợp, Aerospike cố gắng cung cấp tính khả dụng tốt nhất và giảm thiểu các vấn đề có thể xảy ra.

Trong trường hợp lệch đồng hồ (ngoại lệ 1), một giao thức của Aerospike được gọi là ***Aerospike's gossip cluster protocol*** liên tục theo dõi lượng sai lệch thời gian và sẽ cảnh báo nếu độ lệch trở nên lớn - và vô hiệu hóa cụm này trước khi xảy ra mất dữ liệu.

Trong trường hợp server lỗi hoặc bị sập, Aerospike nhanh chóng tự động tái tạo và cân bằng lại dữ liệu trong trường hợp lần lỗi đầu tiên. Nếu lỗi xảy ra nhanh chóng, các phần của dữ liệu có thể được đánh dấu là “dead”, và yêu cầu sự can thiệp của người điều hành. Điều này là để cho phép sử dụng dữ liệu ở chế độ chỉ đọc hoặc cho phép tải lại các thay đổi trước đó .

1. **Cấu hình cho Strong Consistency**

Strong Consistency khả dụng cho toàn bộ một namespace.

Các bước để cấu hình Strong Consistency cho một namespace:

1. Lấy khóa để mở khóa cho cấu hình Strong Consistency.

2. Cài đặt NTP(hoặc đồng hồ đồng bộ hóa khác)

3. Thêm tệp cài đặt để xác định Strong consistency namespace.

4. Cấu hình cho roster ban đầu.

5. Cấu hình node IDs với Strong consistency

6. Cấu hình Rack Awareness

Chi tiết xem thêm: [Tài liệu về quản lý Strong Consistency](https://docs.aerospike.com/docs/operations/configure/consistency/index.html) giải thích cách cấu hình một namespace và một roster.

1. **Quản lý Strong Consistency:**

Quản lý Strong Consistency phức tạp hơn nhiều so với quản lý những namespace có sẵn.

[Tài liệu](https://docs.aerospike.com/docs/operations/manage/consistency/index.html) này mô tả cách thêm và xóa bỏ node, bắt đầu và dừng server một cách an toàn, và nhiều khái niệm quản lý khác

1. **Nâng cấp từ AP(Available and Partition-tolerant) lên SC(Strong Consistency) namespaces:**

Về cơ bản, chuyển từ AP lên SC namespace không được hỗ trợ. Một số trường hợp vi phạm tính nhất quán có thể xảy ra trong quá trình chuyển đổi. Tạo ra một namespace, sau đó sao lưu và khôi phục được đề xuất trong trường hợp này (quá trình này được gọi là: thủ tục nâng cấp “forklift” ).

1. **Sử dụng Strong Consistency:**

Có rất ít thay đổi lập trình viên có thể làm. Chỉ cần biết đơn giản là dữ liệu an toàn.

Tuy nhiên, vẫn có một số thay đổi mà lập trình viên có thể thực hiện chủ yếu được quản lý thông qua client Policy object, cũng như là sự khác biệt trong ý nghĩa của mã lỗi.

1. **Cơ chế đọc khả tuyến tính: (Linearizable Reads)**

Đọc khả tuyến tính là một trường mới tồn tại trong ‘Policy’ object mà lập trình viên có thể thay đổi. Để sử dụng đọc tuyến tính (Linearizable reads), cần phải đặt lại giá trị của trường **linearizableRead** thành **true.** Nếu giá trị trường này được thiết lập để đọc trên một namespace không cấu hình Strong consistency, việc đọc sẽ thất bại. Nếu đặt giá trị này lên một SC namespace, bạn sẽ đạt được Session Consistency.

Policy object có thể được cài đặt mặc định trong hàm khởi tạo của AerospikeClient, như vậy tất cả những lệnh gọi sau đó cũng sẽ mặc định dùng giá trị của Policy object. Ngược lại, ta nên sử dụng hàm khởi tạo Policy object trên từng khởi tạo để xác định rõ thực thi một cơ chế đọc tuần tự toàn phần, hay đọc nhất quán từng phần.

Một số Policy objects như là BatchPolicy, WritePolicy, QueryPolicy và ScanPolicy cũng kế thừa từ Policy object. Trong những Policy này, trường linearizableRead chỉ có nghĩa cho đối tượng mặc định và đối tượng kế thừa là BatchPolicy, được áp dụng cho tất cả các thao tác batch

1. **InDoubt Errors:**

Trường **InDoubt** được thêm vào để xác định sự khác biệt giữa một lệnh ghi *chắc chắn chưa được* thêm vào, hay *có thể* đã được thêm vào.

Đối với hầu hết cơ sở dữ liệu, những lỗi như TIMEOUT là không rõ ràng, ví dụ như khi client gửi dữ liệu tới server nhưng không nhận được phản hồi từ server thì không thể biết được liệu server đã ghi dữ liệu vào cơ sở dữ liệu hay chưa. Tuy nhiên, đối với Aerospike API, khi có lỗi **InDoubt** thì chắc chắn là dữ liệu chưa được ghi vào.

1. **Data Unreachable Errors:**

Đôi khi client không thể truy xuất được dữ liệu, lý do có thể là mạng hoặc các hạ tầng khác. Có 4 lỗi có thể xảy ra khi một cluster bị phân mảnh, hoặc xảy ra lỗi phần cứng dẫn đến dữ liệu không có sẵn (4 lỗi này có mã không giống nhau ở các phiên bản Client khác nhau, dưới đây là mã lỗi được sử dụng trong Java Client): **PARTITION\_UNAVAILABLE**, **INVALID\_NODE\_ERROR**, **TIMEOUT** và **CONNECTION\_ERROR.**

**PARTITION\_UNAVAILABLE:** trả về khi server xác định được cluster không có dữ liệu chính xác.

**INVALID\_NODE\_ERROR:** được tạo ra trên client khi client không có một node cụ thể để gửi yêu cầu đi.

**TIMEOUT** và **CONNECTION\_ERROR:** trả về trong trường hợp xảy ra phân vùng mạng.

1. **Tính khả tuyến tính:**

Tất cả những truy cập gần như là song song với nhau đều là tuần tự. Cơ chế quản lý hiệu quả của Aerospike đảm bảo cho từng bản ghi (record) và khiến một giao tác ngầm kiểm tra tình trạng của các server khác.

Nếu một lệnh ghi được thực hiện và được đọc ra bởi client, sẽ không có bất kì một phiên bản nào trước đó được đọc ra nữa. Khi chế độ ‘global consistency ’ này được bật lên, mỗi client liên kết với một cluster sẽ chỉ thấy cùng một giá trị cho một bản ghi tại một thời điểm.

Chế độ này yêu cầu thêm đồng bộ hóa trên từng lệnh đọc, điều này dẫn đến ảnh hưởng hiệu quả truy xuất.

1. **Nhất quán theo phiên (Session Consistency)**

Chế độ Nhất quán theo phiên là chế độ thực tế nhất trong chế độ Strong Consistency.

Không giống với Linearizable, Nhất quán theo phiên nhắm tới từng phân vùng client, ở đây là một cluster trên một hệ thống client riêng biệt, trừ khi phần bộ nhớ chia sẻ được dùng để chia sẻ trạng thái giữa cluster và các tiến trình.

Nhất quán theo phiên sẽ luôn đảm bảo các giao tác đọc hoặc ghi diễn ra riêng lẻ. Nhất quán theo phiên cung cấp nhất quán có dự đoán cho một phiên, và lưu lượng đọc tối đa mà vẫn đảm bảo độ trễ thấp nhất cho các giao tác đọc và ghi.

1. **Các trường hợp ngoại lệ với đảm bảo nhất quán (Consistency Guarantees)**

Các trường hợp dưới đây có thể dẫn tới một số vấn đề với tính nhất quán hoặc tính bền vững của dữ liệu.

* 1. **Đồng hồ không liên tục( Clock discontinuity)**

Một Clock discontinuity với một khoảng thời gian xấp xỉ hơn 27s có thể dẫn đến mất dữ liệu ghi.

Độ lệch của organic clock không phải là vấn đề, cơ chế quản lí thời gian heartbeat của từng cụm sẽ phát hiện ra độ lệch thời gian là 15s và thực hiện cảnh báo. Nếu độ lệch lớn hơn 20s, node sẽ từ chối lệnh ghi dữ liệu, trả về mã lỗi ‘forbidden’ từ đó ngăn cản vi phạm nhất quán.

Loại lỗi lệch thời gian này thường được chi làm 4 nguyên nhân chính:

1. Administrator Error, khi một người quản trị thực hiện cài đặt khóa tới tương lai hoặc trong quá khứ .

2. Các thành phần đồng bộ hóa thời gian hoạt động sai.

3. Chế độ ngủ đông của máy ảo.

4. Tiến trình Linux tạm dừng, hoặc Docker container tạm dừng.

* 1. **Clock discontinuity do sự hợp nhất máy ảo trực tiếp.**

Trong môi trường máy ảo hoặc môi trường ảo hóa vẫn có thể triển khai nhất quán an toàn.

Tuy nhiên, có 2 vấn đề cấu hình cụ thể là nguy hiểm.

Live migration: là quá trình di chuyển một máy ảo từ một máy vật lý sang một máy khác có thể gây ra nhiều nguy hiểm. Những đồng hồ thời gian di chuyển với sự gián đoạn và bộ đệm mạng có thể duy trì ở mức thấp và được áp dụng vào sau này.

Nếu việc di chuyển trực tiếp là dưới 27s, strong consistency có thể được duy trì. Tuy nhiên, cách an toàn hơn là dừng Aerospike an toàn, sau đó di chuyển máy ảo và khởi động lại Aerospike.

Trường hợp thứ hai liên quan tới tiến trình tạm dừng, có thể xảy ra trong môi trường container. Những tiến trình này gây ra mối nguy tương tự như di chuyển máy ảo trực tiếp, và không nên được thực hiện. Thật ra có rất ít trường hợp cần sử dụng những tiến trình gây lỗi này.

* 1. **UDFs**

Các lần đọc UDF không được tuyến tính hóa và UDF ghi sẽ thất bại trong một số trường hợp sẽ gây ra mất nhất quán.

* 1. **Xóa dữ liệu hết hạn và dữ liệu cần loại bỏ :**

Việc xóa không nhất quán, bao gồm cả xóa dữ liệu cũ hoặc dữ liệu cần loại bỏ cũng gây mất tính nhất quán. Việc xóa đi những dữ liệu này khỏi cơ sở dữ liệu liên tục có thể gây vi phạm đảm bảo tính nhất quán. Trong những trường hợp này có thể thấy dữ liệu trả về. Vì lý do này, thường yêu cầu người dùng tắt tính năng ‘eviction and expiration’ trong cấu hình Strong Consistency.

* 1. **Client retransmit**

Nếu người dùng bật lại tính năng cho phép Aerospike client truyền lại lệnh ghi (retransmission), có thể sẽ có trường hợp gây vi phạm tính nhất quán. Đó là bởi vì một lệnh ghi bị quá thời gian sau đó truyền lại lần nữa, và có nguy cơ được ghi vào cơ sở dữ liệu nhiều lần.

Điều này gây ra nhiều lỗi:

1. dữ liệu được ghi vào nhiều lần gây mất tính nhất quán. Có thể tránh bằng việc sử dụng mẫu “read-modify-write” và chỉ định cho một lệnh cụ thể, cũng như là vô hiệu hóa retransmission

2. Tạo ra các mã lỗi sai. Ví dụ, một lệnh ghi có thể đã được ghi nhưng do lỗi đường truyền khiến cho client gửi lại lỗi ghi. Trong lần ghi thứ 2, bộ nhớ đầy, và tạo ra một lỗi ‘InDoubt’ thậm chí khi giao tác đã hoàn tất thành công. Loại lỗi này có thể được giải quyết bằng cách dùng mẫu ‘read-modify-write’.

* 1. **Secondary Index requests (scan and query) :**

Việc thực thi một truy vấn có thể xảy ra hiện tượng dirty read. `Để đảm bảo về hiệu suất, Aerospike sẽ trả về dữ liệu ở trạng thái chưa được commit.

1. **Ngoại lệ về tính bền vững dữ liệu (Durability Exceptions)** 
   1. **Lỗi lưu trữ phần cứng:**

Khi lỗi xảy ra, phân vùng bộ nhớ bị lỗi sẽ được đánh dấu là **dead\_partition** ở giao diện phía admin.

Lỗi này xảy ra khi tất cả các node trong toàn bộ roster sẵn sàng và đã kết nối, tuy nhiên một số phân vùng dữ liệu chưa sẵn sàng. Để cho phép một phân vùng chấp nhận đọc và ghi trở lại, admin cần ghi đè lỗi bằng cách thực thi một lệnh “revive”, hoặc đưa máy chủ lên trực tuyến với dữ liệu bị thiếu.

* 1. **Quản lý Roster không đúng đắn.**

Giảm số roster bằng cách giảm hệ số nhân bản hoặc nhiều node tại cùng một thời điểm có thể gây ra mất dữ liệu. Quy trình thêm hoặc xóa bỏ an toàn dữ liệu khỏi cluster phải được tuân thủ.

* 1. **Xóa một phần bộ nhớ (Partial storage erasure )**

Trong trường hợp một số lượng sectors hoặc portion của ổ đĩa bị xóa bởi người vận hành, Aerospike sẽ không thể ghi nhận lỗi. Xóa một phần dữ liệu trên vùng nhớ nhân bản hoặc nhiều node có thể gây xóa bản ghi hoặc làm mất phát hiện lỗi.

* 1. **Máy chủ khởi động lại đồng thời:**

Mặc định, Aerospike ghi dữ liệu vào bộ nhớ đệm và xem xét các dữ liệu được ghi khi tất cả các node máy chủ yêu cầu đã nhận được dữ liệu và bỏ dữ liệu vào đúng hàng chờ.

Aerospike sẽ ngay lập tức sao lưu lại dữ liệu khi server xảy ra lỗi. Nếu quá trình sao lưu nhanh chóng, sẽ không có dữ liệu ghi bị mất.

Để ngăn chặn việc mất dữ liệu trong hàng đợi trong suốt quá trình lỗi liên tục, việc bật tính năng **commit-to-device** là cần thiết. Thuật toán mặc định của tính năng này khiến cho dữ liệu mất được hạn chế và vẫn đảm bảo hiệu suất cao. Tính năng này cũng có thể được bật trên từng namespace riêng biệt để đảm bảo mức hiệu suất cao hơn. Trong trường hợp một tệp hệ thống hay thiết bị có bộ đệm ghi được sử dụng như bộ nhớ lưu trữ, **commit-to-device** có thể không ngăn chặn bất kì loại mất mát dữ liệu nào

**Cơ chế bảo mật:**

Ở phiên bản doanh nghiệp của Aerospike có cơ chế kiểm soát truy cập hệ thống gồm nhiều chế độ xác thực. Những chế độ xác thực này có thể sử dụng riêng biệt hoặc đồng thời.

Admin có thể tạo người dùng cho cho server Aerospike, phân quyền, hoặc có thể dựa vào hệ thống LDPA bên ngoài.

Phiên bản doanh nghiệp của Aerospike hỗ trợ các loại xác thực như:

1. **Xác thực dựa trên mật khẩu người dùng nội bộ được xác định trong máy chủ.**

Tài khoản người dùng được tạo ra trong cơ sở dữ liệu Aerospike (người dùng nội bộ) có thể sử dụng xác thực dựa trên mật khẩu (Password-based authentication), hoặc xác thực dựa trên cơ sở hạ tầng khóa công khai (PKI: Public Key Infrastructure ) dựa trên TLS

TLS: Transport layer security: là một hỗ trợ về luân chuyển dữ liệu giữa các kênh khác nhau.

Password-based authentication:

Quản trị viên tạo ra người dùng của một cụm Aerospike (Aerospike cluster) với tài khoản , mật khẩu và có thể có phân quyền cho người dùng. Đây cũng là chế độ xác thực mặc định cho clients.

1. **Xác thực PKI cho người dùng nội bộ, sử dụng chứng chỉ TLS**

Xác thực PKI là một chế độ xác thực khác cho người dùng nội bộ. Người quản trị sẽ tạo ra username và một password ngẫu nhiên cho người dùng. Nếu người dùng bị hạn chế truy cập dựa trên PKI mTLS, người quản trị sẽ không thông báo mật khẩu cho họ.

1. **Xác thực bên ngoài bằng cách sử dụng một máy chủ LDPA**

LDPA là một tính năng của Aerospike phiên bản dành cho doanh nghiệp.

Xác thực LDPA là một loại xác thực bên ngoài, trái ngược với xác thực cục bộ máy chủ cho tài khoản người dùng nội bộ được xác định trong máy chủ.

Một cụm dữ liệu Aerospike cho phép, hỗ trợ xác thực dựa trên một máy chủ LDPA bên ngoài, không sử dụng người dùng nội bộ. Đồng thời tài khoản người dùng nội bộ cũng không thể sử dụng xác thực LDPA.