Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2022, 9(2):151-155

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Multi Thread & Dynamic Configuration for Date Based Processing of per Day Multi Million Data

Siva Sathyanarayana Movva

Email: sivasathya@gmail.com

Abstract Global Payments or Transactions happen in the form of messages for cash transfers, securities, metals, and various other needs generating millions of data daily. Along with the central applications handling the msg transfer end to end numerous other applications are needed to serve the purpose of reconciliation, business intelligence, tracking of payments and other customer specific requirements. The intent of this paper is to provide the high-level approach, advantage, and technical solution of an application requirement where application receives a configuration formatted by Business Intelligence/Marketing team and extraction of the data to be done with /without message body as per the configuration or set of configurations provided using the mechanism of dynamic configuration and Multi-Threaded Implementation.

Keywords *thread, configuration, messages, application, multi-thread, business intelligence*

1. Introduction

Multi-threaded implementation for date-based processing of multi million daily data using dynamic configuration. Dynamic configuration versus static configuration, In the scenario of multi-threaded application, Dynamic configuration here is a JSON record, upon receiving the config extraction is done based on it versus the static configuration where the configuration would have been embedded in the application algorithmic implementation or be given by a file which is picked up during run time.

Advantage: Dynamism is based on a parameter and we can set as needed. As the config changes we can use/manipulate things without restarting the application there by avoiding down time.
Multi threading Implementation where each thread responsible for extraction of data based on dates and when a thread aligns with the date being processed with that of another thread then the old thread is dropped.

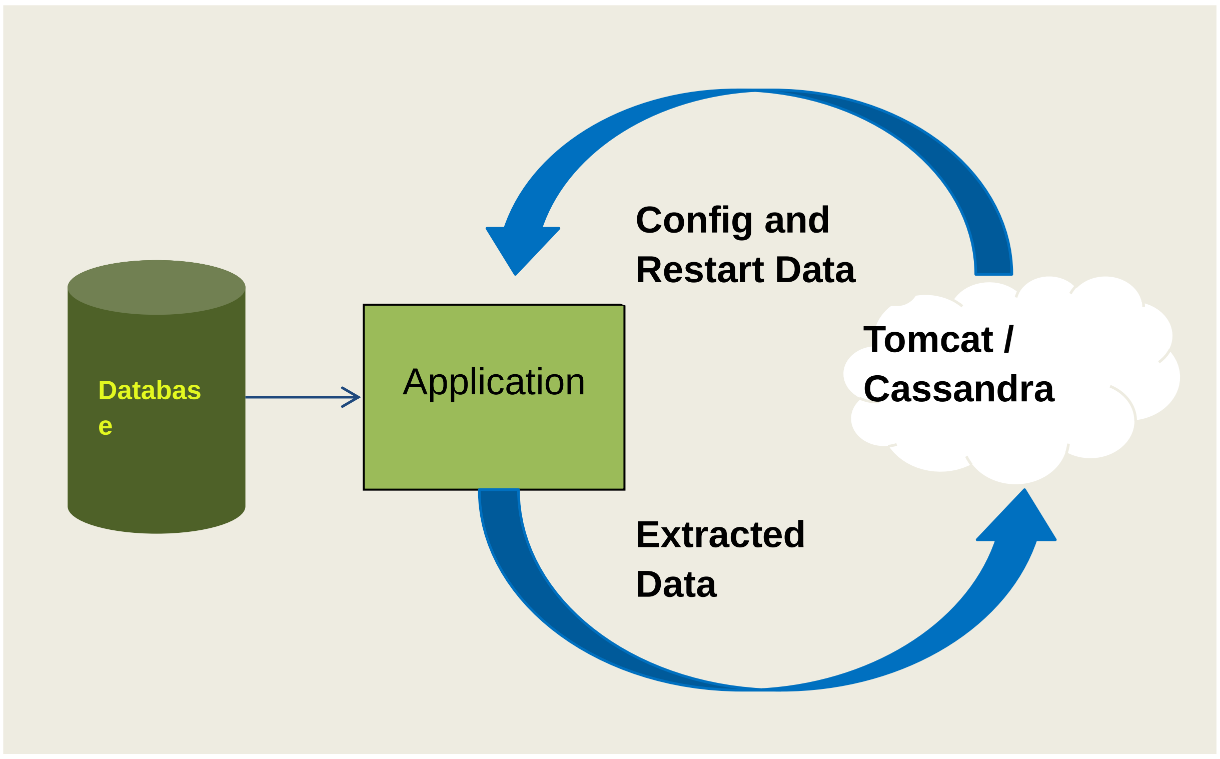
Modality of Dynamic Configuration:

- Queries Configuration and Recovery information from tomcat host.
    - Recovery queried once on start up
    - Configuration is updated periodically through out the day.
    - Configuration changes result in current extract runs being stopped and new extract runs started.
- Data is extracted based on the configuration.
- Multiple extract configurations are processed at the same time.
- Data is batched into JSON objects and sent to a Cassandra server.
- No state is retained on the host.

Parametrized help in implementation:

JSON_SIZE --> Batch size i.e. no of input or output messages sent as a batch to the Tomcat/Cassandra.

HTTP_RETRYCOUNT --> Maximum number of times Recset retries for a http connection with Tomcat/Cassandra.
HTTP_RETRYWAIT --> Maximum time in seconds to wait for http connection in a single retry.
OUT_WAIT_SEC --> When the extraction of input is lagging the output, this param defines how long to wait to check again if input has caught up.
STATUS_POST_TIME --> If the param is set to 10mins then every 10mins the appl info display will get updated.
MAX_LAG --> This param defines how far the processing can lag the current date and time or what's the max time the lag can before event can be generated and alerted.
LAG_EVT_FREQ --> How often that appl will send an event that the processing is lagging.
LAG_ENABLED --> This will enable the mechanism by which we check the appl lag w.r.t. current date and time.
BYPASS_CONFIG_ENABLED --> The param enables a mechanism by which if Output completes before input in a RUN, that particular RUN is bypassed/skipped and the processing continues with the other Runs. In the absence of this mechanism, completion of output before input in any of the RUN's leads to a fatal event bringing the recset down as it is an exceptional condition.



Sample Configuration -

Input:{"ExtractRunInfo":[{"ExtractRunId":"FHR001","ReRunId":"1","AuditVersion":"1"}],"NetworkData":[{" SummaryInfo":{"NETWORK_REFERENCE":"20200616DYEBXXXXEXXX0250000297", "APPLICATION_ID":"F", "SERVICE":"RAM", "REQUESTOR":"DYEBXXXXEXXX", "RESPONDER":"RAMOSG33XXXX", "REQUEST_TYPE":"MT096", "DELIVERY_RESULT":"ACKED", "REJECT_REASON":"Null"},"DetailInfo":{"STANDARDS_VERSION":"0","INPUT_DATETIME":"202006 16224341", "REQUEST_REF":"JLYFUS33A1643495", "IRP_NODE":"105", "MESSAGE_PRIORITY":"S", "DELIVERY_MONITORING":"0", "DELIVERY_WARNING":"0", "ISP_EMITTER":"102", "MUG_PROFILE_ID":"", "TRAINING_MODE":"0", "PDM_FLAG":"0", "IS_SAME_DAY_DELIVERY":"1", "IRP_TIC_VALIDATION":"0", "IS_Y_COPY_MESSAGE":"1", "VASNUM":"113", "VASNUM2":"0", "VASNUM3":"0", "FI_VASNUM1":"0", "FI_VASNUM2":"0", "FI_VASNUM3":"0", "FI_VASNUM4":"0", "FI_VASNUM5":"0", "FI_VASNUM6":"0",

"TFM_SVC1_NUM":"0", "TFM_SVC1_DISP":"0", "TFM_SVC1_DATETIME":"", "TFM_SVC2_NUM":"0", "TFM_SVC2_DISP":"0", "TFM_SVC2_DATETIME":"", "TFM_SVC3_NUM":"0", "TFM_SVC3_DISP":"0", "TFM_SVC3_DATETIME":"", "IS_EMITTER_SYNONYM":"0", "IS_RECEIVER_SYNONYM":"0", "IS_REPLAY":"0", "IS_EMITTER_TRNG_DEST":"0", "IS_SWITCHABLE":"1", "PDM_COUNT":"0", "OBSOLESCENCE_PERIOD":"0", "Y_COPY_ABORT_CODE":"0", "IS_REINSTATED":"0", "MT97_FWD_FLG":"0", "UETR":"", "EMIT_DELTA_MNS":"720"}}],"FINMessage":""}
Output:{"ExtractRunInfo":[{"ExtractRunId":"FHR001","ReRunId":"1","AuditVersion":"1"}],"NetworkData":[{"SummaryInfo":{"NETWORK_REFERENCE":"20200616DYEBXXXXEXXX0250000297","APPLICATION_ID":"F","SERVICE":"RAM","REQUESTOR":"DYEBXXXXEXXX","RESPONDER":"RAMOSG33AXXX","REQUEST_TYPE":"MT096","DELIVERY_RESULT":"DELIVERED","REJECT_REASON":""},"DetailInfo":{"RECORD_TYPE":"1","OSP_NODE":"102","ORP_NODE":"308","HISTORY_LENGTH":"270","OMA_DATETIME":"20200615191031","IS_REPLAYED":"0","O_VASNUM":"113","SYNONYM_OVER_DESTINATION":"0","IRP_DATETIME":"20200616224341","RESPONDER_LT_IDENTIFIER":"RAMOSG33A","MESSAGE_OUTPUT_REFERENCE":"200617RAMOSG33AXXX0040000185","EMITTER_SYNONYM":"AAAAAAAAA","RECEIVER_SYNONYM":"AAAAAAAAA","Y_COPY_STATUS":"Normal","ABORT_MSG_DEST":"AAAAAAAA","NUM_OF_ATTEMPTS":"1","MESSAGE_LENGTH":"578","IS_CANCELLED":"0","TOR_NAK_REASON":"","ORP_DATETIME":"20200616220051","QUEUE_INPUT_DATETIME":"20200616220101","IS_SUSPENDED":"0","GPI_TAG_PRESENT":"0"}}]},

Initialization func:

The Initialization functionality, sets up all the parameters. determines which recset it is, builds the ‘frmhttp’ configuration, Initializes all the global parameters.

The Run function, Runs on a while(true) loop, looks at the configuration from the ORS through the JSON parser. Gets the recovery information, calls the parser and builds a map of all the recovery information. Goes through each Run, and then builds threads to process all the runs. Goes through all the Runs and sees which all Runs are being processed and builds a single thread for all the runs on the same date.

Run Explained …

In a particular scenario, say if 9 runs are on current date then all the 9 runs get a single thread and then say one run is with an older date then it gets a second thread. This thread allocation is done in a similar manner both for Input and Output in different loops. Once all threads are built then it starts the input extracts and output extracts. Once thread allocation is done the main Run thread goes into sleep state and when it wakes up it kills everything and restarts the process again. For the same date when there are multiple runs then all are handled by single thread and the processing starts from the lowest of sequence no.(Configured for different extractors for that date).

Appl Functionality Explained …

InputExtract - Reads all the text messages and determines which ones are to be extracted based on the configuration which ORS has provided.

OutputExtract - Reads all the history messages and determines which ones are to be extracted based on the configuration which ORS has provided.

InputMsgPost - Sends input data to the ORS.

OutputMsgPost - Sends Output data to the ORS.

LoadSVCCodeMap - It loads the ycopy service map. mapping of the ycopy names to their respective nos.

getConfig → This is the function where we go and read the configuration from ORS, the configuration we get is in the form of JSON.

setInputFetchCriteria → Sets up all the info needed for store procedure runs.

setOutputFetchCriteria → Sets up all the info needed for store procedure runs.

checkRunIDAgainstEndDate → Checks the Enddate and sets if the Runs are completed or not.

setRecoveryInfoVec → Sets up all the recovery information.

setOrGetIRPTime, setPostIRPTime, checkPostIRPTime --

- Got to keep track of Irp Times because Output Irp time should always lag the Input Irp Time.

- Irp time of the last text message compared to the last hist message and if lag is found then sending hist to the ORS should be postponed until text catches up.
- Auto lock mutex is being used in input/output post and fetch criteria as multiple threads will be reusing the functionality.
-

Appl Function Explained …

- Total 3 loops to post extracted output and input messages can be seen in the Input extractor and Output Extractor.
- First loop --> To post when maximum amount of data we have to post as per the design is available.
- Second loop --> To post when after being done with 50 iterations and still don't have enough data, we post what we have.
- Third loop --> If we don't have any data to post, then picks up the SCN from DB and posts it along with the jrnl pointers.
-

Process of Introducing new Runs

For any installations on ORS side, or any new extracts configuration on ORS side, the normal process is:
We shutdown the Recovery sets, Shutdown ORS application, Install all necessary changes (ORS changes) on ORS hosts, Once installation is successful, -> we bring up ORS first and then the appl Recovery sets.

Conclusion

Let us summarize the over all flow –
In the case of multi-thread implementation for date based processing of multi-million data and with dynamic configuration, the configuration is provided to application in the form of JSON record. The configuration can have multiple runs, the runs being differentiated by dates, types of messages extracted, fields in the messages being extracted, if the body is extracted or not and many such variants. The extraction starts with the oldest date of all the RUN's with Thread1, and Thread2 with next date and so on. As Thread 1 completes the day, the thread dissipates but provides input to all the Run's which are in need of the data for that day and so on. In the current implementation a total of 10 threads manage the over all process and with a restart mechanism of every 10 hours. This helps in the speedy reconciliation mechanism as 10 threads crunch the over all back dated data at any point of time and also avoids the re-processing of data for a day which is already processed.

References

[1]. Berg, Formal Methods of Program Verification and Specification, Prentice Hall, 1982.
[2]. B. W. Boehm, "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, pp. 61-72, May 1988.
[3]. G. Booch, Object-Oriented Design with Applications, Benjamin Cummings, 1991.
[4]. S. Adve and K. Gharachorloo. Shared memory consistency models: a tutorial. IEEE Computer, 29(12):66–76, Dec. 1996.
[5]. B. L. O. Andersen. Program Analysis and Specialization for the C Programming Language. PhD thesis, DIKU, University of Copenhagen, May 1994.
[6]. S. R. Ladd, Turbo C++ Techniques and Applications, M T Books, 1990.
[7]. S. B. Lippman, C+ + Primer, Addison Wesley, 1991.
[8]. P. J. Lukas, The C+ + Programmers Handbook, Prentice Hall, 1992.
[9]. B. Meyer, Object-Oriented Software Construction, Prentice Hall, 1988.
[10]. A. Aiken and D. Gay. Barrier inference. In Proceedings of the 25th Annual ACM Symposium on the Principles of Programming Languages, Paris, France, Jan. 1998. ACM
[11]. R. R. Seban, *A Temporal Logic for Proofs of Correctness of Distributed Protocols*, March 1993.
[12]. R. R. Seban, *An Introduction to Object-Oriented Design with C++*, December 1992.
[13]. I. Sommerville, Software Engineering, Addison Wesley, 1992.
[14]. B. Stroustrup, The C++ Programming Language, Addison Wesley, 1989.
[15]. R. H. Thayer, "System and Software Requirements Engineering", *IEEE Tutorial*, 1990.

[16]. Callahan, K. Kennedy, and J. Subhlok. Analysis of event synchronization in a parallel programming tool. In Proceedings of the 2nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seattle, WA, Mar. 1990.

[17]. D. Chase, M. Wegman, and F. Zadek. Analysis of pointers and structures. In Proceedings of the SIGPLAN '90 Conference on Program Language Design and Implementation, pages 296–310, White Plains, NY, June 1990. ACM, New York.