國立臺中科技大學資訊工程系碩士班

碩士論文

基於 Apache Spark 建構 XML Veracity 真實度之模型

Veracity Model of Streaming XML Document Based on Apache Spark

指導教授：陳世穎 Shih-Ying Chen

研究生：紀承甫 Cheng-Fu Ji

中華民國 108 年 7 月

基於 Apache Spark 建構 XML Veracity 真實度之模型

Veracity Model of Streaming XML Document Based on Apache Spark

指導教授：陳世穎 Shih-Ying Chen

研究生：紀承甫 Cheng-Fu Ji

國立臺中科技大學
資訊工程系碩士班
碩士論文

A Thesis
Submitted to
Department of Computer Science and Information Engineering
National Taichung University of Science and Technology
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

July 2019

Taichung, Taiwan, Republic of China

中華民國 108 年 7 月

基於 Apache Spark 建構串流 XML Veracity 真實度之模型

學生： 紀承甫　　指導教授： 陳世穎

國立臺中科技大學 資訊工程系

摘要

近年大數據的數量飛速成長，而龐大的數據會產生大量的應用。每一個應用都會產生資料交換，XML (eXtensible Markup Language) 作為現今通用的網路資料交換格式，隨著網際網路資料的增長，也已經同樣具有大數據 (Big Data) 的特徵。本研究建構 XML 真實度模型應用程式開發介面 (XML Veracity Model API) 來解決大數據在資料傳輸中真實度不易量化的問題。XML 文件真實度基於資料理解性有很多面向可以做詮釋，使用真實度模型 API，使用者可以自行設計自己所認為的文件真實度的維度以及屬性，並產生量化的分數。且為了因應現今串流資料應用的增加，且又因串流 XML 資料又有結構上的問題，難以驗證真實度。本研究將真實度模型應用到串流資料，並且使用 Apache Spark 增加模型的處理效能，來達到快速處理串流 XML 的目的，以及驗證真實度模型的設計架構。

關鍵字：巨量資料、XML、串流、Apache Spark、真實度

Streaming XML Veracity Model Based on Apache Spark

Student： Cheng-Fu Ji　　Advisor： Shin-Ying Chen

Department of computer Science and Information Engineering

National Taichung University of Science and Technology

Abstract

In recent years, the volume of data contained by the big data has increased rapidly, and huge data will generate a lot of applications. Every application generates data exchange. XML (Extensible Markup Language) is a common network data exchange format. With the growth of Internet data, it also has the characteristics of big data. This study constructs the XML Veracity application development interface (XML Veracity Model API) to solve the problem that the realism of big data is not easy to quantify in data transmission. The truth of XML file is based on data understandability and can be interpreted and used. Veracity Model API, users can design their own dimensions and attributes of document realism and generate quantified scores. In order to respond to the increase in the application of streaming data today, and because of the structural problems of streaming XML data, it is difficult to verify the realism. This study applies the Veracity model to streaming data and uses Apache Spark to increase the processing performance of the model to achieve the purpose of quickly processing streaming XML and verifying the design architecture of the realism model.

Keywords：Big Data, XML, Streaming, Apache Spark, Veracity

目錄

表目錄

圖目錄

第一章　緒論

1.1　背景

近年來數據以飛快的速度成長，TB 或是 PB 等級的數據隨處可見。在這資料快速產生且數據快速交換的時代，大數據一詞也常被提及。國際數據公司 (International Data Corporation, IDC) 有研究指出 [1]，在 2011 年，創建和複製的資訊量以及資料量將超過 1.8 ZB。這麼龐大的資料也成了產業界與學術界所需要探討的重要議題。而有這麼大量的資料也意味著會有大量的應用產生，而這一些應用當中一定會需要資料的交換，而在交換資料的時候，大多數的應用會選擇 XML。

在大數據中常使用 5V [2] [3] 來描述大數據的特性。所謂的 5V 是指 Volume、Vleocity、Variety,、Value 以及 Veracity。Volume 是指產生的資料量；Velocity 是指資料產生的速度；Variety 是指資料的多樣性；Value 是指資料的價值；Veracity [4] 是指資料的可信度或是真實度。而大數據在做資料交換的時候我們所關心的是資料的可信度或是真實度，也就是上述所提到的 Veracity，這會影響到我們在做資料分析的結果可信度。假使進來的資料不具可信度的話，那麼所分析出來的結果自然也就不具有任何價值。本研究基於 Apache Spark 建構 XML Veacity 真實度模型，利用 Apache Spark 分散式大數據處理引擎，來計算並建構大型 XML 文件的 Veracity 模型，並且將此模型應用在串流 XML 文件的 Veracity 真實度評分。

1.2　XML

可延伸標記式語言 (Extensible Markup Language，簡稱 XML) [5]，是一種作為資料交換的結構化資料格式。XML 是從標準通用標記式語言 (SGML) 中簡化修改出來的。XML 是從 1995 年開始有雛形，並向全球資訊網聯盟 (World Wide Web Consortium，簡稱 W3C) 提案，而在 1998 年 2 月發布為 W3C 的標準 (XML 1.0)。

XML 的特色在於文件內的標籤名稱可以由使用者自行定義，而 XML 文件必須是結構完整的 (well-formed)。所謂的 well-formed 是指 XML 除了要符合每一個標籤都要有起始元素之巢狀結構之外，還要符合格式規範。在 XML 中我們為

了確保文件的格式正確性，會使用 DTD(Document Type Definition) [6] 或是 XML Scheme [7]。DTD 是 XML 提供的文件驗證機制，這是用來定義文件合法區塊，也就是定義元素的架構，有使用 DTD 的 XML 都必須依照 DTD 所定義的格式來呈現。圖 1是一個使用 DTD 且 well-formed 的 XML 文件，在圖 1當中的第 2 行到第 8 行即是 DTD 驗證的格式。從第 2 行開始宣告了這一份 XML 文件的根節點 (root)，再來第 3 行宣告了根節點下有哪一些子節點，第 4 行到第 7 行宣告了子節點下皆為資料，沒有接續的子節點。

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE note[
3    <!ELEMENT note (to, from, heading, body)>
4    <!ELEMENT to (#PCDATA)>
5    <!ELEMENT from (#PCDATA)>
6    <!ELEMENT heading (#PCDATA)>
7    <!ELEMENT body (#PCDATA)>
8  ]>
9  <note>
10   <to> Tove </to>
11   <from> jani </from>
12   <heading> Reminder </heading>
13   <body> Don't␣forget␣me␣this␣weekend!␣</body>
14 </note>
```

圖 1: example1.xml

但由於 XML DTD 並不能完全滿足 XML 自動化處理的要求，也缺乏對於文件結構屬性和資料屬性的規範。所以 W3C 在 2001 年的時候提出 XML Schema。XML Schema 使用的語法與 XML 相同，且支援多種資料類型。圖 2是一個使用 XML Schema 驗證的 XML 文件，在圖 2的第 4 行即宣告了 XML Schema 的路徑位置，而 XML Schema 的格式如圖 3所示，且可以看到在語法的部分與 XML 相同採用巢狀的結構。而 XML Schema 與 DTD 的不同之處在第 5 行到第 10 行，XML Schema 對於每一個節點有著更詳細的資料型態定義，在語法以及文件結構上面也更趨近於 XML。

```
1  <?xml version = "1.0"?>
2  <card xmlns = "http://businesscard.org"
3     xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation = "business.xsd">
5
6     <name> John Doe </name>
7     <title> CEO, Widget Inc. </title>
8     <email> john.doe@widget,com </email>
9     <phone> (202) 456-1414 </phone>
10    <logo url = "widget.gif"/>
11 </card>
```

圖 2: example2.xml

```
1  <schema xmlns = "http://www.w3.org/2001/XMLSchmea"
2      xmlns:b = "http://businesscard.org"
3      targetNamespace = "http://businesscard.org">
4
5    <element name = "card" type = "b:card_type" />
6    <element name = "name" type = "string" />
7    <element name = "title" type = "string" />
8    <element name = "email" type = "string" />
9    <element name = "phone" type = "string" />
10   <element name = "logo" type "b:logo_type" />
11
12   <complexType name = "card_type">
13     <sequence>
14       <element ref = "b:name" />
15       <element ref = "b:title" />
16       <element ref = "b:email" />
17       <element ref = "b:phone" minOccurs = "0" />
18       <element ref = "b:logo" minOccurs = "0" />
19     </sequence>
20   </complexType>
21
22   <complexType name = "logo_type">
23       <attribute name = "url" type = "anyURI" />
24   </complexType>
25 </schema>
```

圖 3: XML Schema

1.3 Apache Spark

為了解決大數據的計算問題，在 2008 年的時候 Google 提出了 MapReduce [8] 的運算框架，用於大數據的處理。MapReduce 是由 Map 和 Reduce 所組成的，Map 的動作是將大的計算任務切割成小任務來操作，Reduce 是將前面 Map 切出來並算完的小任務做合併，來取得最終的結果。後來這樣的一個計算框架演變成我們現在所熟悉的 Hadoop [9]。

在 2010 年 Yahoo 提出了 HDFS(Hadoop Distributed File System) [10] 來解決大數據資料儲存的問題。HDFS 為一個分散式檔案儲存系統，將單一檔案切割成數份，分別複製以及儲存到叢集節點當中，以達到儲存的目的。而這樣的的系統好處有三點：第一點為可擴展性 (Data Scalability)，當叢集的儲存空間不足的時候可以非常輕易的做擴充；第二點為容錯性 (Fauit Tolerance)：當其中一個儲存節點內資料有損壞的時候，系統可以從其他節點找到遺失的資料切片的備份來還原資料本身；第三點為並行性 (High Concurrency)：藉由分散式檔案系統可以讓資料並行處理。然而 Hadoop 有一個缺點，在進行 MapReduce 的時候，每一次的計算都是需要硬碟的 I/O。這樣的行爲造成了效能的瓶頸。所以有了新一代的運算框架。

Apache Spark [11] 為一個開源的大數據分散式引擎，最初是 2009 年由加州大學柏克萊分校 AMPLab 所提出，為記憶體內 (In-memory) 的計算。跟 Hadoop 相比，Apache Spark 的計算效率比起 Hadoop 來說有顯著的提升，其原因為 Apache Spark 在運算的時候，將運算中間產生的資料暫存在記憶體中，因此可以加快整體運行速度，而 Hadoop 則是在每一次計算完成或是產生中間資料的時候，都必須對硬碟動作，這個動作降低了 Hadoop 的執行效率，而 Apache Spark 的設計則能夠增加計算的效率。除了上述提到的效能問題之外，Hadoop 只支援 Map 和 Reduce 這兩種運算，對於現今要處理的資料來說，在撰寫程式的時候靈活度不夠，而且 MapReduce 在運行任務的時候，任務排程以及啟動開銷大，基於上述原因，Apache Spark 是目前較好的大數據處理引擎。

Apache Spark 主要的對資料的操作是使用叫做 RDD(Resilient Distributed Datasets，彈性分佈資料集) [12]，RDD 的結構如圖 4所示。RDD 是具有容錯機制以及高效

能的抽象資料結構。在圖 4中黃色的區塊是 RDD 當中所謂的 Partition，是指資料的分片。一個 RDD 會由一個到多個的 Partition 所組成，程式在運行的時候，Parttion 會分散至各叢集節點當中進行運算，會存放在記憶體內，所以可以快速分享各個 Partition 的運算結果。RDD 是一個可以並列操作且有容錯機制的資料集合。且可以透過參照外部儲存系統的資料集建立，或者是透過現有的 RDD 轉換而創建，例如 map, join, reduce 等對於資料的操作，而在 Apache Spark 對於 RDD 的操作中，有分為轉換 (Transformation) 和動作 (Action)，Apache Spark 在做資料操作的時候，採用的是惰性求值，也就是當 Apache Spark 在做 Transformation 的時候，並不是馬上會做資料的轉換，而是會先把要轉換的動作記錄下來，等到有呼叫 Action 的 API 的時候才會依照剛才做資料的操作並輸出結果，這樣可以使執行效能提高，例如一個資料經過 MapReduce 處理後會得到一個結果，而不是返回一個大的資料集。

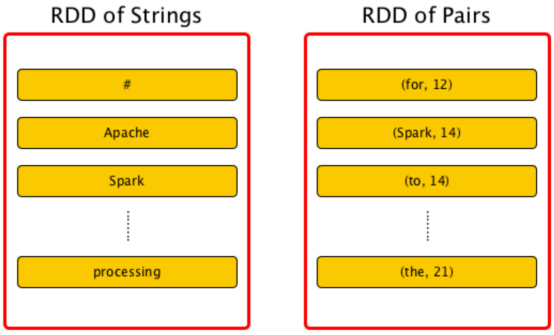

圖 4: RDD 結構

Apache Spark 有著豐富的函數庫，也針對 Python, Java, Scala, R 等程式語言提供一致的 API，架構如圖 5，而 Apache Spark 當中的核心的為 Spark Core，所有 API 都以此為基礎。Spark Core 提供了分散式任務調度、排程和基本 I/O ，而主要的程式抽象化結構 RDD 也是定義在這邊，RDD 抽象化是經由 Apache Spark 所支援的程式語言整合 API 呈現的，簡化了寫程式的複雜性。基於 Spark Core，Apache Spark 中提供了 Spark SQL、Spark Streaing、MLlib 以及 GraphX 5 大類 API 供使用者進行呼叫。

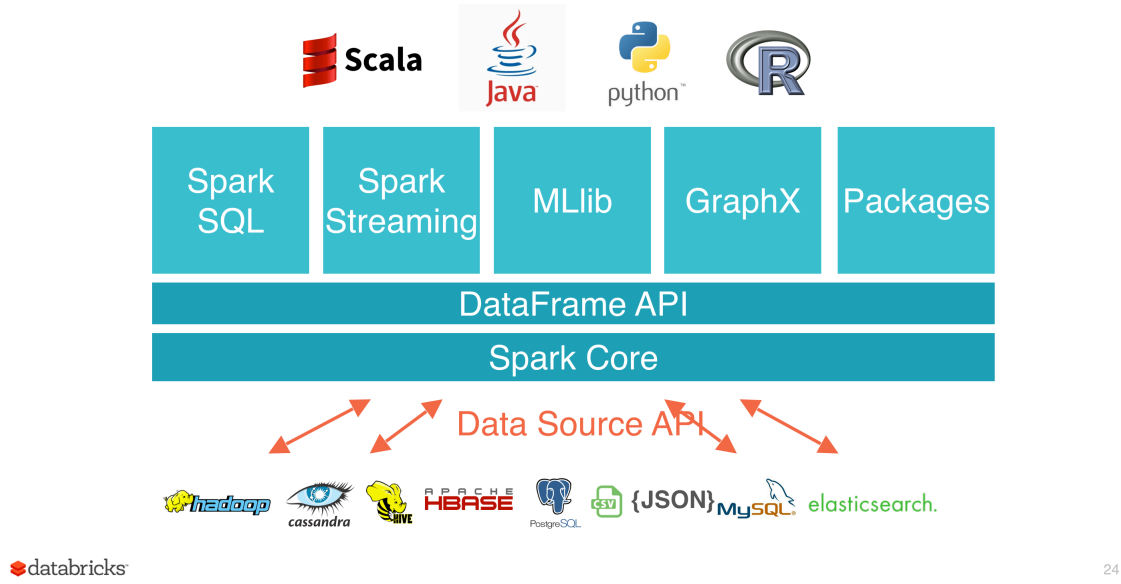

圖 5: Apache Spark 核心架構

Spark SQL 在 Spark Core 上定義了一個叫做 SchemaRDD 的資料抽象化概念，提供結構化和半結構化資料的相關支援，另外 Spark SQL 允許使用者使用 SQL 遠法做資料查詢，也允許程式開發人員將 SQL 查詢與其他 RDD 所支援的資料處理方式一起使用。Spark Streaming 是在處理即時串流資料的元件，例如 web server 所產生的紀錄，或是服務狀態的改變，Spark Streaming 擷取了微批次 (Micro Batch) 的資料並對資料執行 RDD 的轉換。所謂的微批次是指將每次處理時間的間隔縮小到數秒內，甚至是毫秒等級，雖然也是批次處理，但由於時間間隔變短，感覺起來會趨近於即時處理的效果。Spark Streaming 的處理是針對每一個微批次做資料操作，如圖 6所示：

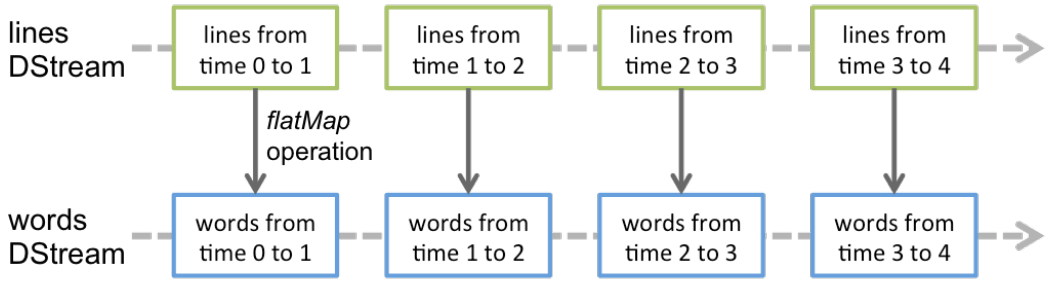

圖 6: Spark Streaming 微批次資料操作

MLlib 是 Apache Spark 所提供的機器學習 (Machine Learning) 函數庫，MLlib 可以使用許多常見的機器學習和統計學的演算法，大幅度簡化了機器學習的時間，其中提供了匯總統計、分類、回歸、分群、協同過濾以及最佳化等機器學習及統計分析的演算法。

GraphX 是 Apache Spark 上的分散式圖形處理框架，此框架可用於表達圖形計算並可以類比 Pregel 抽象化，GraphX 提供了許多處理圖像的操作，例如 subgraph 和 mapVertices 以及常見的圖形演算法。

1.4 大數據 XML 文件

所謂的大數據 [3] 是指無法以人工管理或是處理的資料集，也可以定義為多來源的結構化或非結構化的資料，而在大數據當中，我們常常用”5V” 來描述大數據的特性，這 5V 分別是規模性 (Volume)、價值 (Value)、真實性 (Veracity)、時效性 (Velocity) 及多樣性 (Variety)。

5V 當中，Volume 所代表的是大數據的規模，也就是大數據以傳統的儲存方式已經無法負荷此龐大的資料量或是傳統的資料處理方式已無法對其做運算了；Value 是指大數據透過處理後所得到的資料產生的價值；Veracity 是指資料的品質，可信度，資料是否可靠，或是結構的完整性；Velocity 是指大數據資料產生的速度；Variety 是指大數據的資料多樣性，在設計有關大數據的應用的時候，藉由上面五個的特性有助於檢視大數據的特徵。

XML 文件的 Veracity 真實度這個特性是會有不同的變化的。例如 XML 文件如果以是否符合 DTD 規範，well-formed 以及 vaild 的 XML 文件在真實度 Veracity 的特性上面就會有不同的變化。我們以兩份 well-formed XML 來說，上述有提到 XML 文件的標籤是可以讓使用者自行定義的，所以會造成兩份文件出現相同標籤名稱，意思不同，或是不同標籤名稱，意思相同的狀況，會造成不易判斷的狀況，也有可能在解析完 DTD 和 Schema 後發現兩份文件的結構很相似，所以真實度很高的狀況。

從資料理解性 (data understandability) 的角度詮釋 Veracity 的意義來看，假設有兩份 XML 文件，一份為基準文件 B，一份為被量測文件 M；我們可以從多個角

度去描述「以基準文件觀之，這份被量測文件多少可能是真實的」。例如文件 M 與文件來源相同，因此文件 M 應該是真實的。或是文件 M 與文件 B 有相同的 DTD，因此文件M應該是真實的。或是文件 M 與文件 B 的 DataGuides 相同，因此文件M應該是真實的。這些對於來源相同、DTD 相同和 DataGuides 相同的「想法」，就是用所謂的資料理解性來描述被量測文件 M 是否是真實的。而由於這樣的想法可以多元且其重要性可能有差異，因此真實性就可以有程度上的區別

而在這 5V 當中，本研究要探討的是真實度模型在 XML 文件的建模與應用，在 [13] 有提出大數據可以由兩個面向來討論，一個是資料理解性，一個是大數據的評價基準，而本研究針對資料理解性面向進行建模，假設有 n 份基準 XML 文件以及 m 份被測量 XML 文件，使用者可以從多個角度對於 XML 文件進行真實度計分，XML 文件真實度包含的面向極廣，對於真實度的定義每一個人不盡相同，所以本研究建構一個在 Apache Spark 叢集上的 Veracity 真實度模型，使用物件導向程式設計 (Object-Oriented Programming，簡稱 OOP) 的觀念，設計一個維度、屬性以及評分的抽象類別，將基本架構定義完整，做成 Veracity Model API 且是跑在 Apache Spark 叢集上，將實作細節交給使用者來決定，前面有提到使用者可以從不同的角度來評價 XML 的真實度評分，以及決定真實度要有多少維度以及多少屬性。並且本研究的模型應用在串流 XML 上，有別於以往傳統批次文件的處理方式，串流的資料處理方式更符合現代的資料處理架構及場景，而串流 XML 文件相較於傳統批次處理的 XML 文件處理方式又有所不同，針對一個大數據串流 XML 真實度評價模型是一個除了評價資料真實度之外，也可以篩選真實度較高之資料的解決方案。

1.5 研究動機

目前在產界與學界中，大數據已經是熱烈討論的議題，而當中的對於 5V 的研究也已經行之有年。本研究對於大數據的 XML 文件有以下分析：在 Volume 的部分因 XML 時常用於網路資料交換，而隨著大數據應用越來越多。資料交換的頻率以及資料量也會越來越大。而大量的應用也使資料產生的速度加快，是屬於 Velocity 的部分。資料來源多樣化以及格式上的多樣化則是屬於 Variety 的部分。大數據當中藏有很多隱藏資訊，藉由資料探勘等技術發現資料的隱藏價值是屬於 Value 的部分。最後是資料的真實度或是可信度因大數據的資料量極大，使用者或

是開發者並不曉得這樣的資料是否可以使用或是否有造假資料參雜其中。

在上述 5V 的分析當中，Veracity 是本研究關心的焦點，因為所有的資料如果真實度或是可信度不高，在儲存上因為儲存了一堆的無價值資料增加伺服器負擔，分析應用上的結果因資料是真實程度或是可信程度低所以造成結果不具任何價值。本研究將模型應用在串流 XML 上，以解決串流 XML 資料因在串流當中結構的不確定性，而導致真實度驗證困難的問題。

1.6 論文架構

本論文其餘架構如下。第二章相關研究將介紹 XML 在分散式系統的處理難處以及問題點，以及在以前是如何做真實度比較。第三章將介紹真實度模型的理論以及設計方法。第四章會實際使用物件導向設計一個真實度模型，以及介紹設計理念。第五章為使用 Apache Spark 在串流 XML 的環境之下間模型實作至叢集當中，並且測試 Apache Spark 對於模型的計算效能以及傳輸效能。第六章將針對模型提出結語以及未來工作方向。

第二章　相關研究

本研究建構一個在 Apache Spark 叢集上的 Veracity 真實度模型，且應用於串流 XML 的真實度計算，而以往的真實度研究在大部分的文獻中都是討論文件相似度。再來 XML 具有樹狀結構以及自描述 (self-desc 特性，所以在分散式系統處理 XML 的時候，如何切割文件，但依然保有 XML 文件樹狀結構和父子節點的關係，以及在 Hadoop 或是 Apache Spark 的分散式架構下做 XML 的 Query。本章節將就有關這些議題的文獻來做討論。

2.1　XML 文件特性

XML 結構與網頁使用的 HTML 十分相似，而兩者最大的不同為 XML 的設計為用來做資料傳輸，而 HTML 的設計是用來呈現資料。再者 XML 雖然可以自行定義 Tag 名稱，但有嚴格的巢狀結構規定。而 HTML 雖然在 Tag 的命名上沒有那麼自由，但有一些 Tag 沒有遵守巢狀結構的規定卻依然可以作動。

在 [14] [15] 當中，針對 XML 的特性描述提到，XML 可以表示成樹狀結構以及 XML 具有自描述 (self-descriptive) 的特性，以及 XML 可以使用如 DTD 或是 Schema 來規範其內容結構。

在自描述性中，XML 可以使用標籤描述資料內容，如圖 7所示，文件裡面的 Tag 具有描述資料的功能，XML 不負責呈現資料，所以程式開發者需要另外撰寫程式來完成。

```
1 <?xml version = "1.0"?>
2 <note>
3   <to>Tove</to>
4   <from>Jani</from>
5   <heading>Reminder</heading>
6   <body>Don't␣forget␣me␣this␣weekend!</body>
7 </note>
```

圖 7: XML 自描述範例

而 XML 的缺點在於需要用 Tag 來存放資料，以文字檔來說容量會比較大，但

以現今的網路傳輸速度以及資料壓縮的技術，這也不再是問題。

2.2 XML 文件的平行化處理

Hongjie et al. 的文獻 [16] 提到將大型 XML 切割成小型的樹狀結構，存進分散式檔案系統，等到使用者的 query 進來之後，再將切割好的資料提取出來，使用 MapReduce 進行查詢。利用 Hadoop 的分散式系統，來到平行化 query。文獻當中使用分散式檔案系統儲存 XML 文件，也就是說文件在儲存之前需要經過切割。文章當中他們是採用自己設計的切割演算法，將大型 XML 文件的樹狀結構切割成小型樹狀結構，接著在使用者的 query 進入的時候，會平行化的對這一些切割出來的小型樹狀結構作查詢。

這裡面有幾個問題點，第一點是當有多個 XML 文件的時候，每一份文件都會切割演算法做切割並且儲存，這時候會產生很多小文件，如何對應切割出來的小文件與原始大文件的關係，這會影響到要對哪一個文件進行處理，而文件切割和對應的部分，在 [17] 當中有提到，一般在 Hadoop 當中，透過 MapReduce 切割 XML 文件的時候，開始標籤 (<tag>) 與結束標籤 (</tag>) 之間的關係會被切割到不同的部分，導致 XML 文件的節點關係不清楚，第二點是文件切割與平行化的問題，在 [16] 當中是使用 Hadoop 做處理，Hadoop 可以自行決定任務的平行化數量。而如何計算和得知切割 XML 的個數與平行化任務的數量各為多少是比較好的，這是在做平行化運算要面臨的問題點。

可以看到圖 8中紅框標示的部分即為 XML 文件切割完之後要進行平行化運算的部分。當中我們比較關注的是 XML 文件怎麼切割？被切割了幾份？以及平行畫的時候會產生的任務數量以及計算量，都是我們要考量的問題。

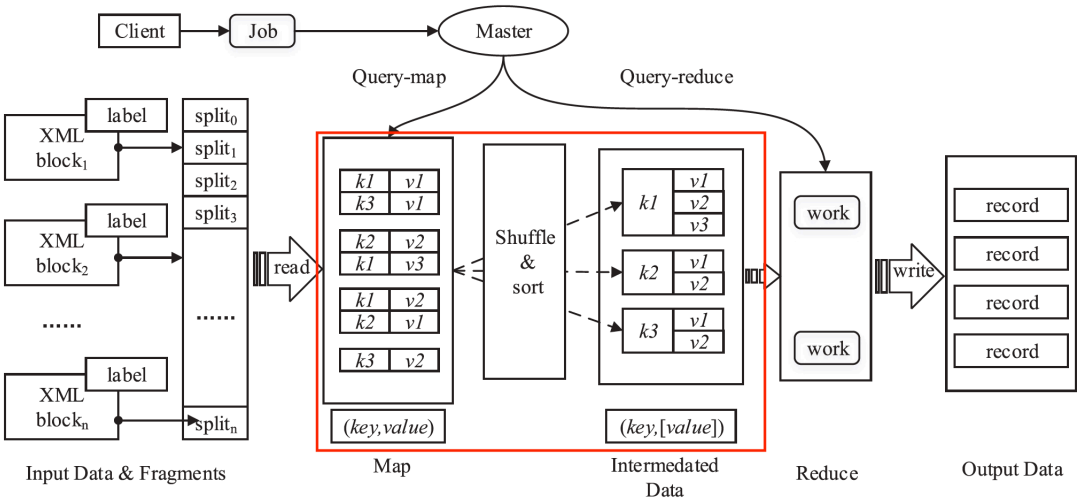

圖 8: XML 在 Hadoop 的切割與平行化

2.3 XML 文件相似度

在本研究所提出的 Veraicty 之問題在以往的相關研究當中算少數，大部分探討是以 XML 文件相似度為主。在以往的研究當中會使用樹的編輯距離 [18](Edit Distance，或稱 Levenshtein Distance) 作相似度的比較，所謂的編輯距離為給定 XML 文件 A 與 XML 文件 B，如果要使文件 B 變成跟文件 A 相同，需要做多少次新增、刪除以及修改的動作。在 [19] 當中即是採用此方法驗證兩份文件的相似度。

而在 [20] 以及 [21] 當中為了增進效能，使用的是子樹 (sub-tree) 來做相似度的計算。而兩篇文獻不同的點在於 [20] 使用的是編輯距離，而 [21] 是對於子樹的結構以及語意相似的子樹出現次數或重複次數進行計算。[22] 則提出了基於樹編輯距離的 XML 語法 (XML grammar) 相似度驗證，並且一樣使用編輯距離來比較 XML 以及 XML 語法的相似度。

接者在 [23] 當中則採用了多種的驗證方式，如 Tag 名稱相似度、編輯距離、N-grams 距離、Tag 語意驗證等。而對於編輯距離的驗證方式除了常見的 Levenshtein Distance 還有使用 Jaro Similarity 的相似度驗證。Jaro 為兩個字串的相似度的度量，如果 Jaro 值越高，代表相似度越高。而另一個則為 N-gram 相似度，作法為將兩組欲比較之字串按照長度 N 切割，則可以計算兩個字串相同的子字串有多少，進而比較原字串相似度。

上述的相關研究大部分是使用編輯距離相似度演算法來做 XML 相似度的問題。而考量到計算效能上的問題，有幾篇的研究採用了子樹的計算來降低計算量進而強化效能。然而這樣的相似度比較對於真實度來說只是其中一個維度。而且無論是哪一種方法，在現今龐大的資料量以及串流數據的場景，皆很難完成即時性的處理。

第三章　Veracity 真實度模型

現今大數據的興起，資料科學也是風行全球。然而在做資料分析的時候，大家最關心的是資料的可性度，也就是大數據 5V 當中的 Veracity。而建構在資料理解性上的真實度時，因為每一個人在看資料的時候，所看的面向是不同的，所以這就造成的標準不同。所以本研究提出 Veracity 真實度模型來提出一個量化的方式解決這個問題。

3.1　模型理論

真實度模型是建構在資料理解性 (data understandability) 上，而所謂的資料理解性不像一般常見的對於 XML 文件相似度的比較。例如要說這兩份資料的真實度很高，也許大家從不同角度觀察，結果都不盡相同。這樣評量資料真實度就顯得很抽象，本研究即提供一個量化的方式，也就是真實度模型來針對 XML 做分數量化，讓使用者自行設計量化方法，來決定如何把抽象化為具體的評比分數。

而在真實度模型 (Veracity Model) 的架構之下會有多個維度 (Dimension)，在維度之下會有的多個屬性 (Attribute)，每一個屬性之下都有其量化的方法 (Quantification)。假設有一份基準 XML 文件 B(Base XML Document) 以及被量測文件 M(Measure XML Document)，而文件 M 將被量測與文件 B 的真實程度 (Veracity Value)。這樣就可以計算出文件 M 在真實度模型中的真實分數。而每一個維度也可以根據下面所屬的屬性計算出維度的分數 (Dimension Degree)，也可以得到維度下方每一個屬性的分數 (Score)。

真實度模型 (M) 的架構當中，會有多個真實維度來表示使用者理解資料的多個面向。每一個維度會描述使用者對於資料理解性的每一個面向。下面定義了一個模型有 n 個維度，D_i 表示模型當中的第 n 個維度：

$$M = \{D_i\}\ ,\ i = 1\ to\ n$$

在真實維度 D_i 當中會有多個真實屬性來描述真實維度的細節。舉例來說，使用者認為資料來源可視為一個維度，那麼這個維度的屬性可能會有來源網址、文件編

碼等等，這樣可以定義成真實維度 D_i 當中的 $A_{i,j}$ 個屬性：

$$D_i = \{A_{i,j}\}\ ,\ j = 1\ to\ n$$

在真實屬性 $A_{i,j}$ 當中，會有其量化的方法 $Q_{i,j}$。讓使用者可以針對每一項屬性去描述如何量化以及在量化的時候可能需要做的正規化。

綜合以上，我們可以從每一個維度的真實屬性中的量化方式得到屬性的分數。且根據不同屬性做權重 $w_{i,j}$ 的調整，再自定義函數 $F_A()$ 來計算所有屬性與權重，進而得到維度的分數：

$$|D_i| = F_A(w_{i,j},\ |Q_{i,j}|)$$

而從上述所得到的維度分數與權重 W_i 做調整，並且放入自定義函數 $F_D()$ 中即可得到整體真實度模型的分數：

$$|M| = F_D(W_i,\ |D_i|)$$

3.2 UML 表達方式

基於上述理論模型，使用 UML 建構模型之物件導向關係圖。真實度模型使用 UML 來表示如圖 9所示。在圖 9當中有 Model、Dimension 和 Attribute 這三個抽象類別。在設計的時候，每一個類別都必須繼承這三個類別的其中一個，並且實作內容。實做出來的子類別則會產生組合的關係。如 Model 下面會有多個 Dimension；Dimension 下則會有多個 Attribute，且三個類別互相相依，當 Dimension 下沒有 Attribute 的時候，這個 Dimension 將不存在；而當 Model 下沒有 Dimension，則這個 Model 也不存在。

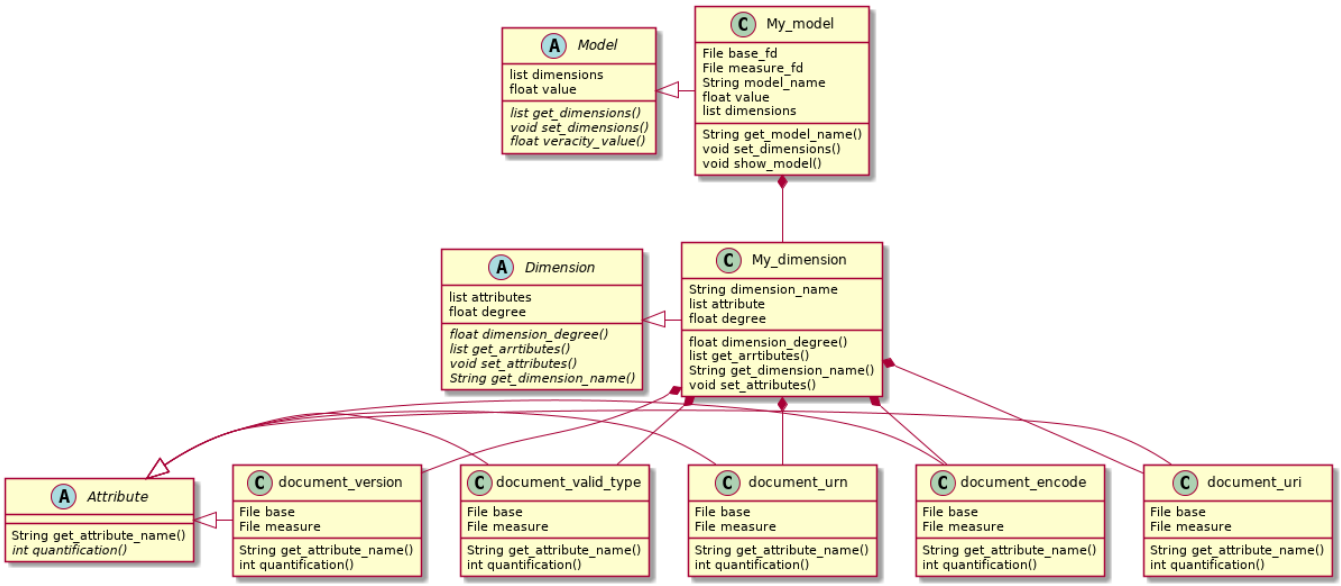

圖 9: 真實度模型

上述之 UML 提供一個物件導向的程式架構，把前面的理論模型轉換為可程式化的方法。也就是只要有這樣的模型以及敘述，使用任一種程式語言皆可以實作本研究之模型，進而達到使用者在選擇程式語言實現本模型時候的彈性。

在實現本模型的時候，需要注意的是抽象類別的繼承與改寫以及物件的組合關係。首先是抽象類別的繼承與改寫，使用者須繼承抽象類別並進行實作。假設進行了繼承卻沒有實作，那麼程式將無法正常動作。再來是物件的組合關係，前面說到三個類別是具有相依關係的。也就是在實作的時候，每一個 Dimension 下至少要有一個 Attribute，這樣這個 Dimension 才得以存在。而每一個 Model 下至少要有一個 Dimension，這樣 Model 才算成立。

第四章　真實度模型實作

本研究使用 OOP 的觀念與特性在 Apache Spark 上面實做真實度模型，在前面有提到對於資料的真實度評價可以從很多面向來做探討，換句話說，藉由 OOP 的設計方式可以讓使用者實作出真實度需要包含哪一些屬性，還有真實度的量化方法、權重加成等方法。

本研究將使用 Python 作為我們設計與實作真實度模型的程式語言。原因是 Python 不但是物件導向的程式語言，而且那它也是 Apache Spark 所支援的程式語言。再來 Python 是十分易懂且容易學習的程式語言，這樣使用者在理解本模型的實作架構的時候，困難度將大大降低。

前面 XML 文件真實度的理論模型的描述方式是由上往下 (Top-Down) 的描述本研究模型的架構，而這樣描述模型的架構，在轉換成程式實作的時候，會無法使用程式語言描述清楚，所以在實作的時候必須採用堆疊式的方法來建構模型，也就是所謂的 Bottom-Up 的設計方法，本研究在設計 API 的時侯先將所有屬性設定完成，接著在維度陣列當中添加設計好的屬性，接著在模型陣列當中添加分類好的維度，依這樣的步驟，由下而上的把模型堆疊上去，這樣的模型是有彈性的，因為在要決定屬性隸屬於哪一個維度的時候，如果有別的模型要使用相同的屬性，那只需要把那一個屬性的物件加入到別的模型的維度陣列當中即可，這樣就可以達到相同的屬性量化方式，但是應用到不同模型維度當中，兼具了物件導向當中的彈性設計以及程式碼的重用性。

4.1　模型實作

真實度模型的建立，其中最重要的就是彈性以及程式的重用性，也就是用物件導向程式設計的特性來達到程式碼的重用，這在於模型是使用者決定他所認為的兩個文件真實度所具備的屬性，也就是說每個人所認為的真實度不同，在實作上就會有所不同，本研究設計真實度模型的抽象類別，首先對於 XML 文件真實度而言，設計一個真實度模型 (Veracity Model)，模型當中會具有多個維度 (Dimension $D_i, i = 1\ to\ n$)，在每一個維度之下會有多個屬性 (Attribute $A_{i,j}, j = 1\ to\ n$) 來描述此維度，並且每一個維度需要有一個量化方法，對應到物件導向上可以將模型視

為一個抽象類別，維度視為一個抽象類別，屬性視為一個抽象類別，所以程式上設計了 Model、Dimension、Attribute 三個主要的抽象類別讓使用者去繼承並實作。

如前面提到因為一個 XML 文件的真實度是可以有很多面向，換言之就是可以從很多維度去探討，維度裡面會有很多屬性，這一些屬性都會有一個對於 XML 真實度的量化方式。

API 的最上層是 Model，Model 底下會有真實度模型的各項元件。而 Python 本身沒有支援抽象類別的設計，所以要使用 Python 設計抽象類別就必須載入一個叫做 abc 的模組，並且在想要抽象的方法前面加入前綴，才能實現抽象類別的設計。整體抽象類別的設計如圖 10所示，在 Model 當中，第 7 行設有 dimension 的 list，用於建立此模型的維度，使用者必須將設計好的 1 到 n 個維度傳入這個 list 當中儲存，利用物件導向的特性，當 Dimension 的物件與 Model 的物件產生後，可以將多個 Dimension 物件也就是多個維度使用第 11 行 set_dimension() 的方法儲存到 dimension 這個 list 當中，接著在第 15 行設計 get_dimension() 的方法來取得這個模型下的維度 list。而最後需要計算模型的整體分數 Veracity Value，所以在第 19 行設計有 veracity_value() 的方法，用以計算模型的整體分數。

```
1  #!/usr/bin/env python3
2  #-*- coding: utf-8 -*-
3  import abc
4
5  class Model(abc.ABC):
6
7      dimension = list()
8      value = 0.0
9
10     @abc.abstractmethod
11     def set_dimensions():
12         pass
13
14     @abc.abstractmethod
15     def get_dimensions():
16         pass
17
18     @abc.abstractmethod
19     def veracity_value():
20         pass
```

圖 10: Model 抽象類別設計

在模型之下會有多個維度，需建立維度的類別來描述維度的架構，維度抽象類別的設計如圖 11所示，使用第 11 行 set_attributes() 將建立好的屬性物件傳入 attributes 的 list 當中存起來，這樣就可以用物件導向的的概念去存取到下面屬性的物件的方法。另外實作上要求使用者在繼承維度的時候設定維度名稱，並且可以用第 15 行的 get_dimension_name() 取得維度名稱。接著需要實作第 19 行 get_attributes() 的方法來取得先前傳入的屬性，在維度當中也會有該維度的計分，所以在第 23 行 dimension_degree() 會設計有維度的計分方法讓使用者實作。

```
1  #!/usr/bin/env python3
2  #-*- coding: utf-8 -*-
3  import abc
4
5  class Dimension(abc.ABC):
6
7      attributes = list()
8      degree = 0.0
9
10     @abc.abstractmethod
11     def set_attributes():
12         pass
13
14     @abc.abstractmethod
15     def get_dimension_name():
16         pass
17
18     @abc.abstractmethod
19     def get_attributes():
20         pass
21
22     @abc.abstractmethod
23     def dimension_degree():
24         pass
```

圖 11: Dimension 抽象類別設計

而在屬性抽象類別當中，設計有兩個方法，分別是 get_attribute_name() 以及 quantification()，抽象類別設計如圖 12所示，使用者需繼承這一個類別來實作屬性的量化方式，在 get_attribute_name() 中本實作直接設計這個方法將回傳類別名稱，也就是要求開發者在繼承並且實作的時候，類別名可以直接取成相關的屬性名稱，這樣的設計方式也是為了在存取屬性的時候方便識別。

```
1  #!/usr/bin/env python3
2  #-*- coding: utf-8 -*-
3  import abc
4
5  class Attribute(abc.ABC):
6
7      def get_attribute_name(self):
8          return __class__.__name__
9
10     @abc.abstractmethod
11     def quantification():
12         pass
```

圖 12: Attribute 抽象類別設計

以上 Model、Dimension、Attribute 三大類別構成了真實度模型的基礎架構，模型的建立由使用者繼承維這三大類別，使用者須先實作完屬性的量化方式接著產生屬性物件並添加至維度之內，接者再將維度物件添加至模型當中，並且在實作每一個類別的時候，須將抽象類別當中的量化方法實作完畢，接著在模型中計算總體量化的分數。

第五章　實驗

本研究將真實度模型放在 Apache Spark 叢集當中運行，並且實際應用在串流 XML 文件上面。因串流資料講求即時接收即時處理，所以難在短時間內做真實度驗證。所以此模型使用在串流資料的環境下可以解決這樣的問題。本研究利用 XML 文件產生器 [24] 來產生 XML 文件，且每一個文件的大小皆不相同，實驗會使用客戶端程式隨機挑選不同大小的文件進行上傳至串流伺服器，再由伺服器分別做資料前處理以及真實度計分，並且產生報表。

因為 XML 通常用於資料交換，所以單一檔案不會到 GB 等級，所以串流 XML 文件在本研究的設定會是小容量且持續產生的檔案流。

5.1　真實度模型之串流 XML 文件應用架構

本研究設計並實作真實度模型在 Apache Spark 叢集當中。Apache Spark 作為真實度計分伺服器，將接收來自客戶端傳送過來的資料，客戶端會使用 socket 將每一份 XML 文件上傳至伺服器端。在 Apache Spark 是使用 Spark Streaming 當中的 Socket streaming 來進行接收。而在 Socket streaming 當中會將接收到的資料進行處理並且儲存，再由真實度模型進行計分，以下將針對系統當中的傳送端、處理模組以及真實度計分模組做介紹。

在資料傳送端，本研究自行實作 Socket 客戶端進行資料的上傳，每一個檔案會以字串的形式來做上傳，也就是說實際上客戶端發送的是持續產生的字串流，整體發送的時序圖如圖 13所示：

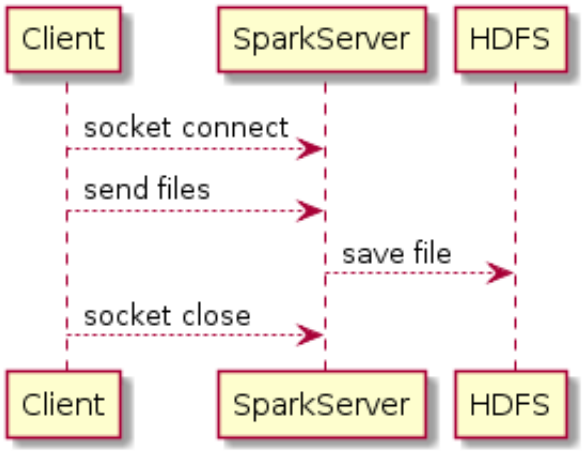

圖 13: 資料接收與儲存流程

在伺服器的處理模組使用的是 Spark Streaming 來接收串流資料，接收到串流資料的時候因資料是以字串流的方式發送的，所以資料處理模組會將字串流切分出每一個 XML 檔案，並且儲存至 HDFS 中。資料傳送處理流程如圖 14所示：

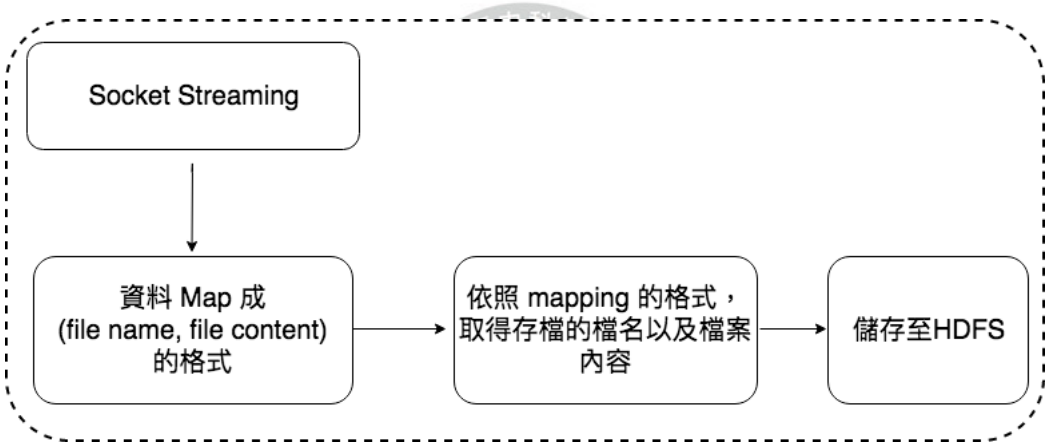

圖 14: 資料傳輸處理模組

在資料儲存的同時，真實度模型即開始從 HDFS 進行檔案的讀取，將傳送進來的 XML 文件與系統當中的基準文件做真實度計分，流程如圖 15所示：

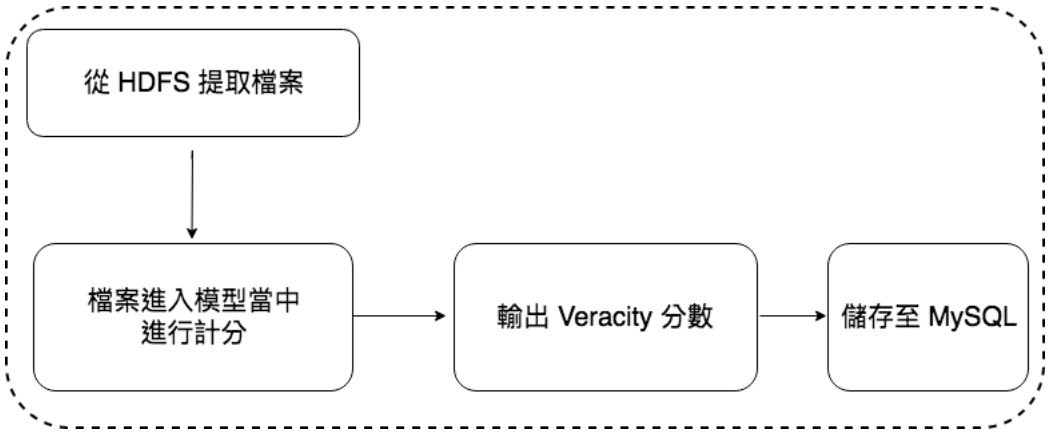

圖 15: 資料真實度計分模組

整體系統由傳送端、資料處理模組以及真實度模型計分模組所構成。利用串流做資料的上傳，從串流資料當中分割出每一個檔案來做儲存，並且使用真實度模型給予每一個檔案做真實度計分。完整系統架構如圖 16所示：

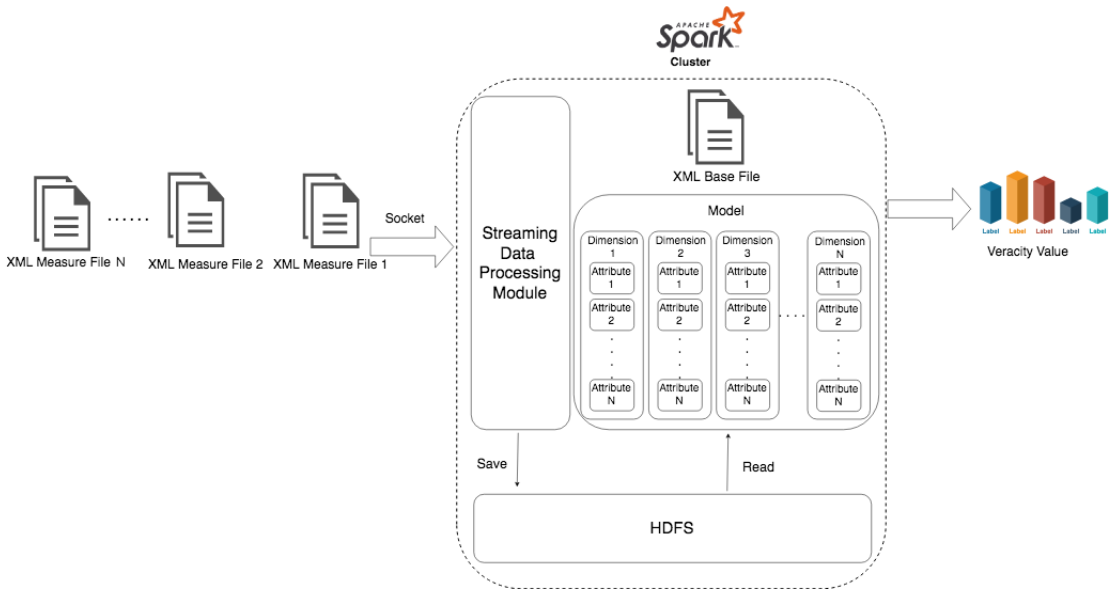

圖 16: 系統架構圖

5.2 真實度模型之應用實例

本研究設計之真實度模型 API 應用在前面所提到的串流 XML 文件上，藉由實驗，驗證真實度模型應用於串流 XML 文件的可行性。而由於真實度的計算方式可彈性由使用者自行設計，所以根據前面的抽象類別，在真實度模型的設計應用上，本實驗設計了兩個維度的真實度模型作為計算 XML 真實度的應用範例。

實驗設計的 XML 資料真實度應用範例具有兩個維度，分別是 XML 文件宣告以及 XML 文件結構。文件宣告的維度當中有文件編碼、文件版本以及是否為 standalone 等 3 個屬性，文件結構的維度當中有文件深度以及是否使用 DTD 或是 Scheme 等 2 個屬性。圖 17為設計的模型架構：

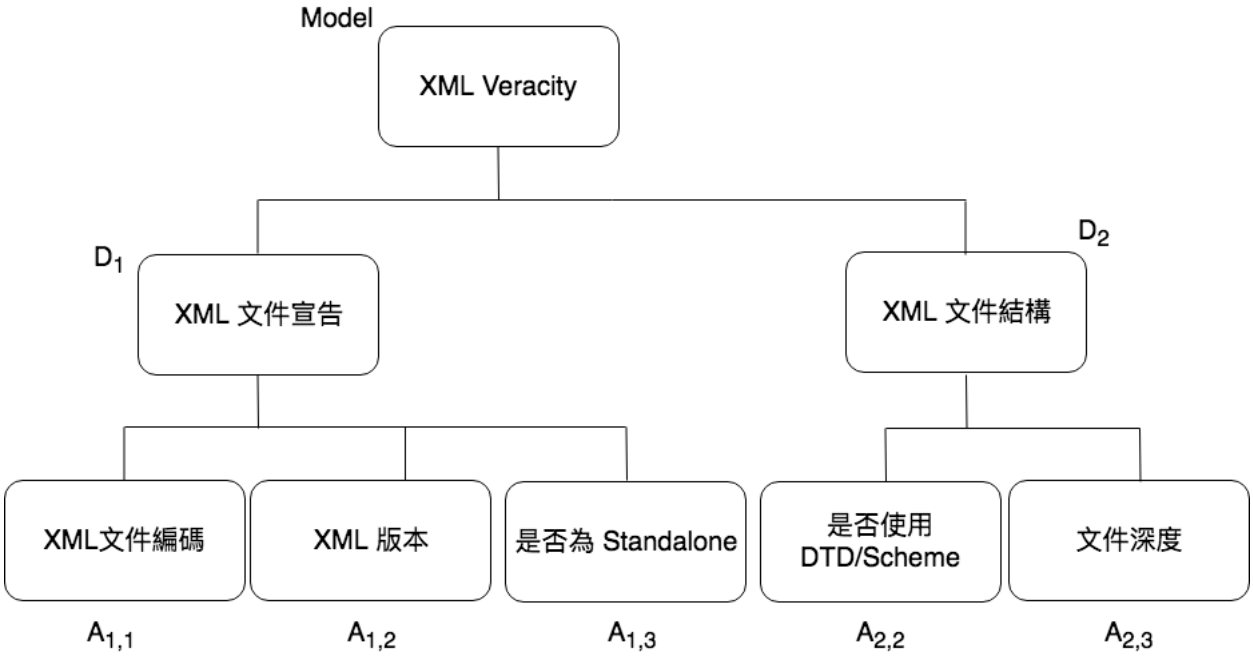

圖 17: 真實度模型

這樣的架構可以對應到前面所提到的理論模型，在維度方面的定義 D_1 為 XML 文件宣告，D_2 為 XML 文件結構：

$$M = \{D_1,\ D_2\}$$

在 D_1 維度下會有 3 個屬性 $A_{1,1}$、$A_{1,2}$ 和 $A_{1,3}$，$A_{1,1}$ 為 XML 文件編碼，$A_{1,2}$ 為

XML 版本，$A_{1,3}$ 為是否該文件為獨立存在：

$$D_1 = \{A_{1,1},\ A_{1,2},\ A_{1,3}\}$$

而在 D_2 維度下會有三個屬性 $A_{2,1}$ 以及 $A_{2,2}$。而屬性 $A_{2,1}$ 代表文件深度，$A_{2,2}$ 代表是否使用 DTD/Scheme：

$$D_2 = \{A_{2,1},\ A_{2,2}\}$$

理論模型確立之後，即可以依照這樣的架構來實作模型，模型建立的演算法如圖18所示：

```
1  def main():
2      mod1 = model()
3      mod1.model_name = "XML Veracity Model"
4      d1 = dimension("XML 文件宣告")
5      d2 = dimension("XML 文件結構")
6
7      mod1.set_dimensions(d1)
8      mod1.set_dimensions(d2)
9
10     d1.set_attributes(document_version())
11     d1.set_attributes(document_encode())
12     d1.set_attributes(document_standalone())
13
14     d2.set_attributes(document_vaild())
15     d2.set_attributes(document_depth())
```

圖 18: 模型建立演算法

接著需要設計每一個屬性都會有的一個計分方式 (Quantification)。依文件宣告的維度來說，下面的兩個屬性：文件版本以及文件編碼，在文件版本來說，如果被量測文件與基準文件的版本不相同，那麼系統在分數的判定上會較為低分。如被測量文件與基準文件的版本是相同的，那系統的評分會是滿分。倘若在基準文件有做編碼宣告，但被量測文件並未做編碼宣告，那此項屬性的系統評分會是 0 分

而在文件編碼的部分，倘若被量測文件與基準文件採用相同編碼，則系統評分為滿分，如果不相同，但因為被量測文件還是有做編碼宣告，所以系統評分會是 60 分及格。如果基準文件有做編碼宣告而被量測文件沒有編碼宣告，此項屬性之

系統評分則為 0 分。

再來是文件有無宣告 standalone，standalone 的宣告代表有沒有引用外部 DTD，如果值為 no，那就代表該文件的 DTD 是宣告在文件外部，如果值為 yes，那代表 DTD 的宣告是在文件內部。如果被量測文件與基準文件的值同樣為 yes 或是 no，那麼系統的評分會是滿分。若是被量測文件有宣告，但是裡面的值跟基準文件不同，則分數會稍微低一些。若是被量測文件無宣告 standalone，屬性的系統評分為 0 分。

在文件結構的維度下會有兩個屬性：有無使用 DTD/Scheme 以及文件深度。在有無使用 DTD/Scheme 的部分，假設被量測文件與基準文件所使用的驗證方式相同，那麼系統計分會給高分，如果不相同，但是基於被量測文件仍然有使用驗證，所以系統會給及格分。假設被量測文件沒有使用任何驗證方式，那系統的屬性計分會給零分。

在文件深度的部分，實驗中以 Dataguides [25] [26] [27] 來進行計算，所謂的 Dataguides 是一種將 XML 的樹狀結構進行精簡化，來加速 XML 查詢以及節點走訪的速度，並且還可以保留各節點之間的關係。所以文件深度這一個屬性利用了 Dataguides 的特性來進行基準文件以及被量測文件的樹深度計算。如果兩棵樹深度相同。在系統計分上會為滿分，如果被量測文件的樹深度與基準文件相差 2 以內，則分數會照著比例增加。比方說基準文件深度為 5，如果被量測文件的深度為 6 或是 4，那分數的就為加 20%。如果深度的差距到達了 2 層以上，系統即認為被量測文件與基準文件的資訊差異過大，所以會以比例開始扣分。

5.3 實驗環境

實驗環境是使用建置在 Openstack 上面的虛擬機，共使用三個虛擬機。作業系統為 Linux 發行版 Ubuntu 16.04 LTS，使用之 CPU 皆為 8 核心，記憶體皆為 16GB，表 1為機器的詳細規格。使用之 Hadoop 版本為 2.7，Spark 版本為 2.0。在 HDFS 中我們將每一個區塊的資料副本設置為 3，軟體環境配置如表 2。

表 1: 叢集電腦規格

	Master 1 台	Slave 2 台
OS	Ubuntu 16.04 LTS	Ubuntu 16.04 LTS
CPU	Intel Core i7 8700@2.93GHz	Intel(R) Xeon(R) CPU E3-1230 v3 @ 3.30GHz
核心數	8	8
Memory	32 GB	12 GB
儲存空間	500 GB	1 TB

表 2: 使用軟體版本

軟體名稱	版本號
Hadoop	2.7 版
Apache Spark	2.0 版
Python	3.6 版

5.4 實驗設計

實驗的設計主要分成三個部分，第一部分是自行設計的 client 端的傳送效率，在實驗中 client 端會隨機挑選數量不等的檔案進行上傳至 server 端，以此來測試每一個檔案的平均上傳時間。第二個部分為測試處理模組的效能，本實驗在測試從 client 接收資料到處理完畢存檔，每一個檔案的平均處理時間。最後一部分的實驗則是使用 profiling 的工具來測試本研究設計之真實度模型的效能。

第一部分的實驗將隨機從 client 挑選 500 個、700 個以及 900 個檔案做發送，測試檔案的平均發送時間。第二部分的實驗將從 client 接收串流檔案，數量分別是 500 個、700 個以及 900 個來做處理。並且將伺服器端設定成每 3 秒接收一次串流，來測試每一個微批次的處理時間。最後一部分為真實度模型實作在 Apache Spark 中並且使用 profiling 的工具來測試效能。測試資料會從前次傳送的檔案中選擇檔案容量最大以及最小的檔案，藉由效能測試工具來檢視真實度模型的計算能力，並且從報表以及圖表中了解效能瓶頸。

5.5 實驗結果

在實驗結果方面，第一部分要看的是 client 端送效率，client 端會從本地隨機挑選 500 個、700 個以及 900 個檔案大小均不同的檔案，來看傳送速率。圖 19當中橫軸為每次傳送的檔案數量，縱軸為每一次傳送的平均時間，單位是秒。

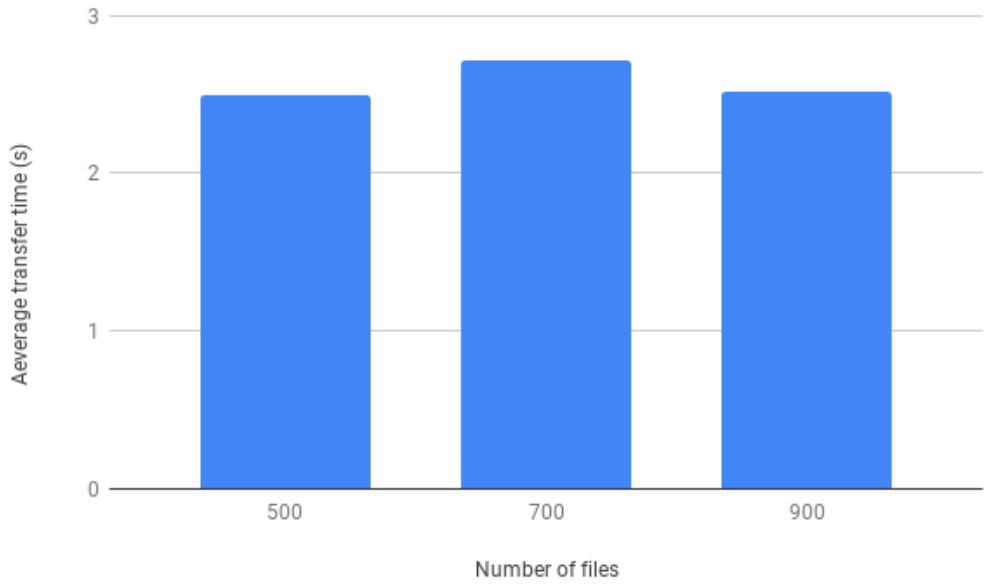

圖 19: client 傳送速率實驗

表 3則是這次實驗的詳細數據。可以看到每一次的檔案傳送在時間上都是十分接近的，三次實驗的時間平均值約為 2.57 秒。

表 3: client 傳送速率實驗詳細數據

檔案數量	平均傳送時間	檔案平均大小	最大傳送檔案	最小傳送檔案
500	2.49 秒	28.57MB	57.91 MB	0.17 MB
700	2.71 秒	28.05MB	57.91MB	0.14 MB
900	2.51 秒	29.14MB	58.09 MB	0.14 MB

第二部分的實驗則是 server 的處理模組在接收到串流資料的平均處理時間，這次的實驗會接收 500 個、700 個以及 900 個檔案大小均不同的檔案，並且計算每一個檔案的平均處理時間。圖 20為三次實驗之統計數據，橫軸為檔案數量，縱軸為單一檔案平均處理時間。

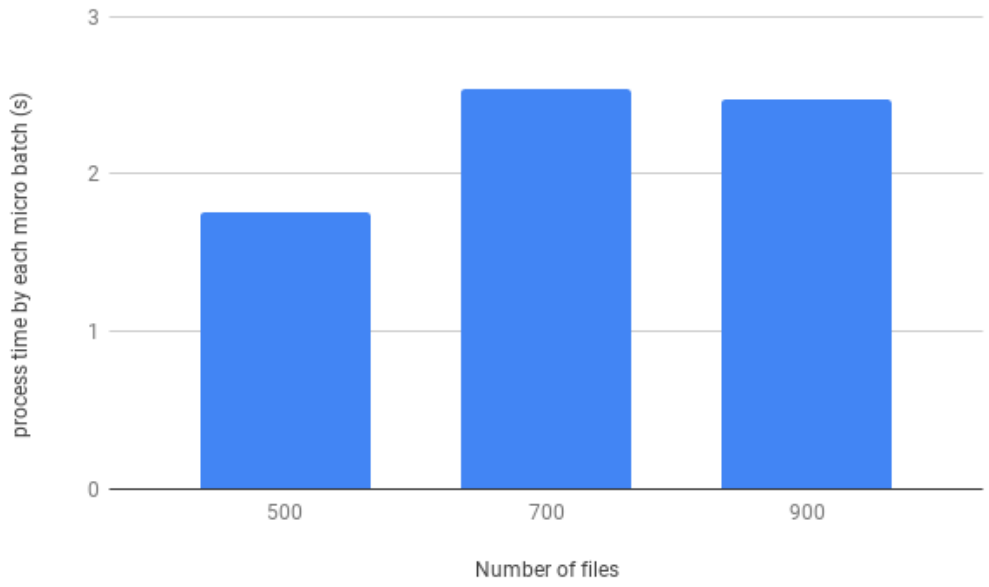

圖 20: 處理模組之處理速度實驗

表 4為此次實驗之詳細數據，可以發現處理模組的處理速度平均值約在 2.254 秒左右，另外可以觀察到的是在檔案數量提升到了 700 與 900 的時候，處理速度上也提升到了單一檔案平均 2.4 秒以上。

表 4: server 端處理模組處理速率實驗詳細數據

檔案數量	單一檔案平均處理時間
500	1.754 秒
700	2.536 秒
900	2.472 秒

最後一部分的實驗為真實度模型實作在 Apache Spark 中並且使用 profiling 的工具來測試效能。本次實驗使用的效能量測工具為 cProfile [28]，cProfile 為 C 語言

實作的 Python 效能量測工具，藉由這項工具可以量測出程式當中每一個函數的執行時間以及呼叫次數等相關資料。本次實驗挑選了五個檔案，詳細檔案數據如表 5、表 6以及表 7所示，這 5 個檔案分別會進行 profiling 的測試，以找出真實度模型的效能問題點。

表 5: 傳送 500 個檔案當中的最大檔案以及最小檔案

檔案名稱	檔案大小
doc10.xml	0.17 MB
doc5216.xml	57.91 MB

表 6: 傳送 700 個檔案當中的最大檔案以及最小檔案

檔案名稱	檔案大小
doc4.xml	0.14 MB
doc5216.xml	57.91 MB

表 7: 傳送 900 個檔案當中的最大檔案以及最小檔案

檔案名稱	檔案大小
doc3.xml	0.14 MB
doc5223.xml	58.09 MB

首先先針對表 5的檔案進行 profiling，並且將結果視覺化，如圖 21和圖 22所示。在 doc10.xml 的實驗當中，總計算時間為 8.980 秒，最花費效能的是 main:30 的函數，消耗了 29.23% 的時間。其次是 main:93 的函數，消耗了 14.08% 的時間，詳細數據如表 8所示。而在 doc5216xml 的實驗當中總計算時間為 49.719 秒，最花費效能的是 main:93 的函數，消耗了 34.19% 的時間。其次是 main:30 的函數，消耗了 16.36% 的時間，詳細數據如表 9所示。

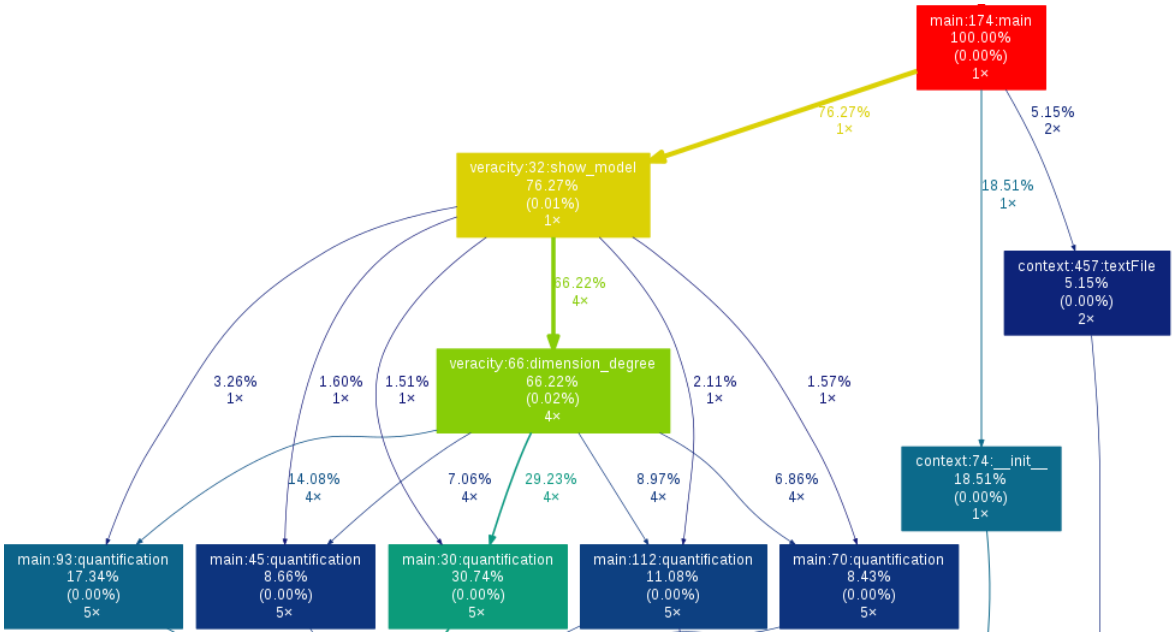

圖 21: doc10.xml profiling 效能分析

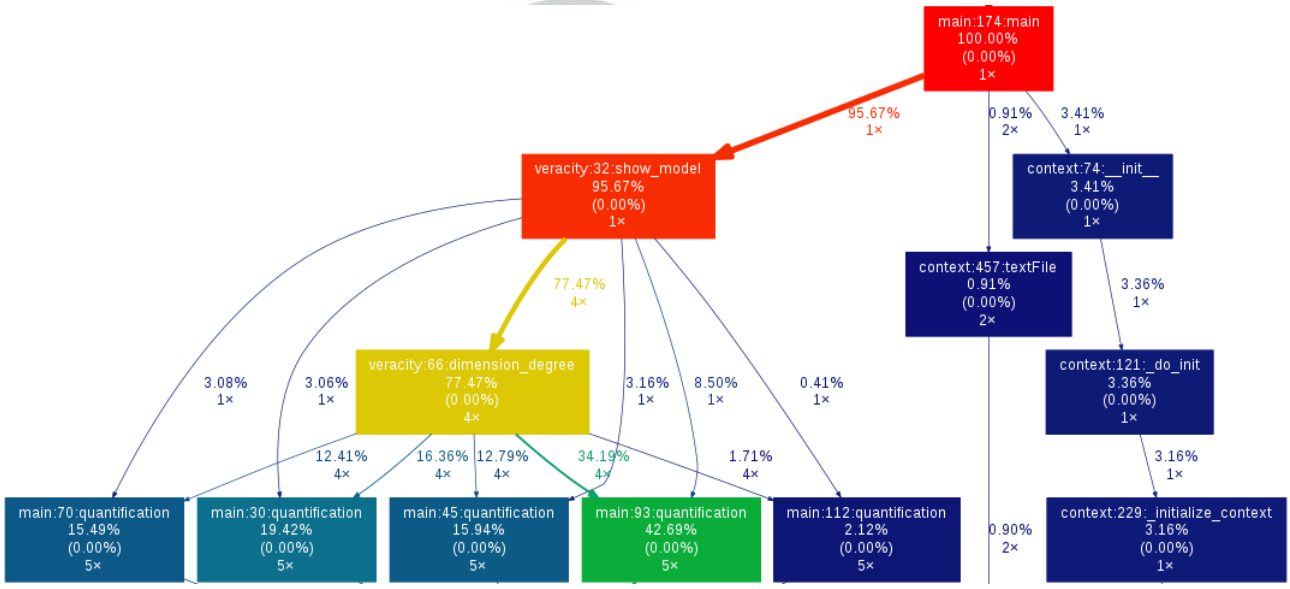

圖 22: doc5216.xml profiling 效能分析

表 8: doc10.xml 計算效能

物件名稱	消耗效能 (%)
document_version	30.74
document_encoding	8.66
document_standalone	8.43
document_valid	17.34
document_depth	11.08

表 9: doc5216.xml 計算效能

物件名稱	消耗效能 (%)
document_version	19.42
document_encoding	15.94
document_standalone	15..49
document_valid	42.69
document_depth	2.12

第二個針對表 6做 profiling 的分析，profiling 的視覺圖如圖 23以及圖 24所示。在 doc4.xml 的 profiling 分析當中，總計算時間為 9.412 秒，最消耗效能的是 main:30 的函數。而在 doc5116.xml 的總計算時間為 50.856 秒，當中最消耗效能的是 main:93 的函數。詳細實驗數據如表 10和表 11所示。

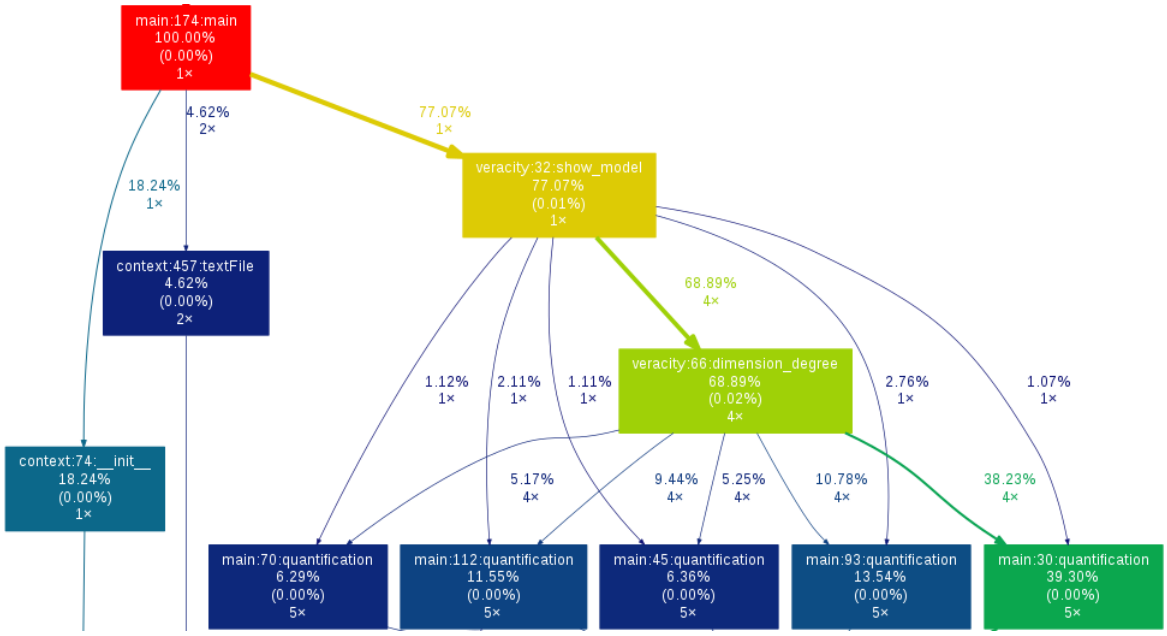

圖 23: doc4.xml profiling 效能分析

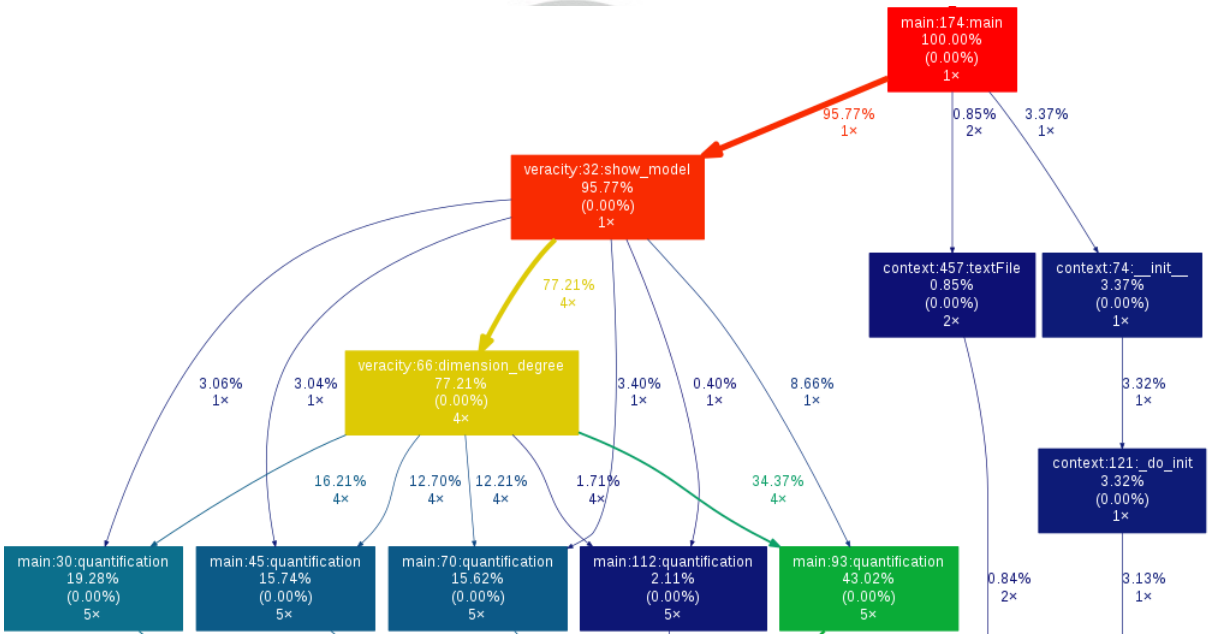

圖 24: doc5216.xml profiling 效能分析

表 10: doc4.xml 計算效能

物件名稱	消耗效能 (%)
document_version	39.30
document_encoding	6.36
document_standalone	6.29
document_valid	13.54
document_depth	11.55

表 11: doc5216.xml 計算效能

物件名稱	消耗效能 (%)
document_version	19.42
document_encoding	15.94
document_standalone	15..49
document_valid	42.69
document_depth	2.12

最後來看到表 7的 profiling 數據，視覺圖如圖 25以及圖 26所示。在 doc3.xml 的 profiling 當中，總計算時間為 8.185 秒，消耗最高的是 main:30 的函數。而在 doc5223.xml 的 profiling 當中，總計算時間為 52.259 秒，消耗最高的函數為 main: 93。詳細實驗數據如表 12與表 13所示。

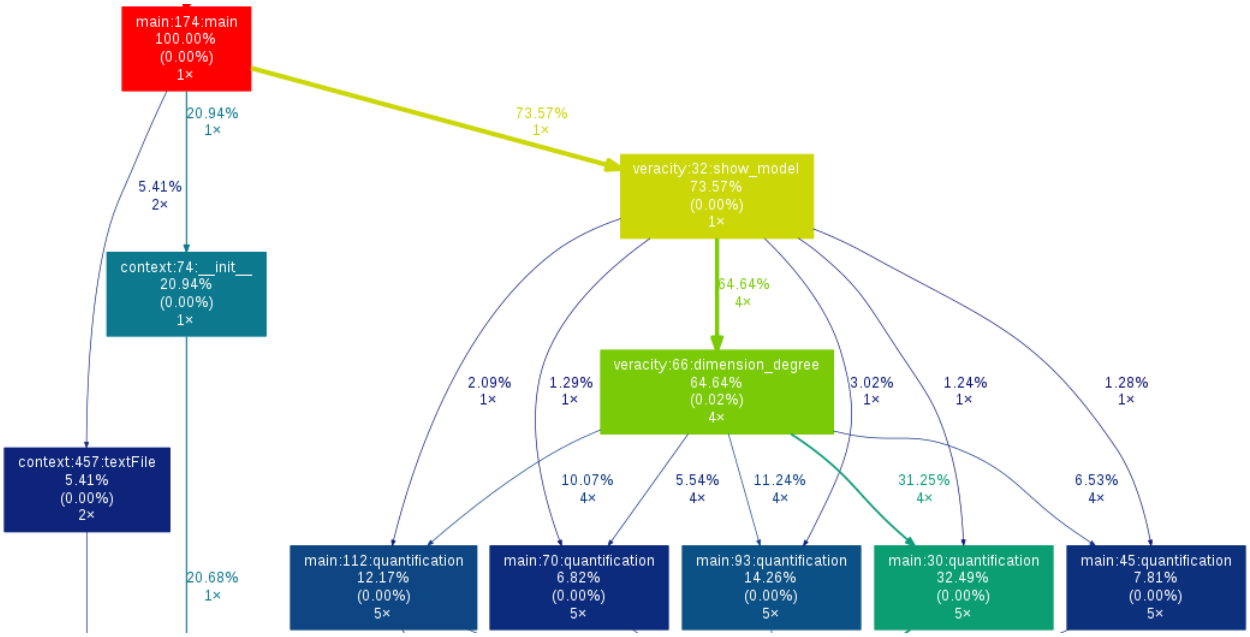

圖 25: doc3.xml profiling 效能分析

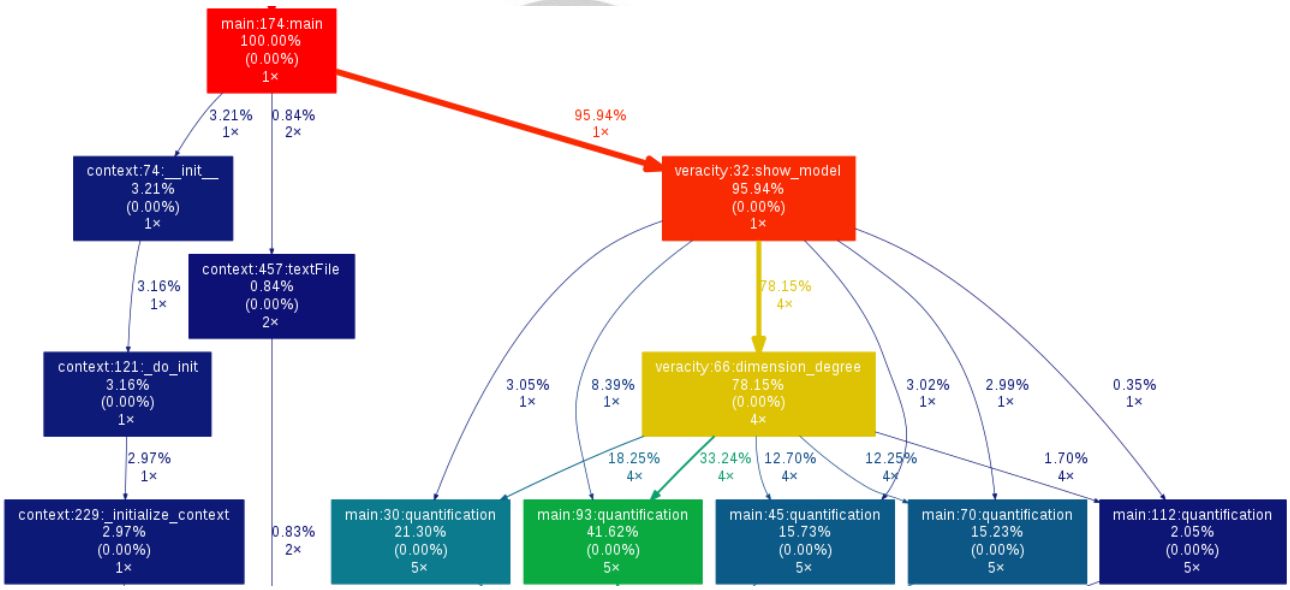

圖 26: doc5223.xml profiling 效能分析

表 12: doc3.xml 計算效能

物件名稱	消耗效能 (%)
document_version	32.49
document_encoding	7.81
document_standalone	6.82
document_valid	14.26
document_depth	12.17

表 13: doc5223.xml 計算效能

物件名稱	消耗效能 (%)
document_version	21.30
document_encoding	15.73
document_standalone	15.23
document_valid	41.62
document_depth	2.05

綜合上 profiling 的實驗結果可以觀察到，在處理大檔案的時候，document_valid() 的屬性會是效能瓶頸。而在處理小檔案的時候反而是 document_version() 會是效能瓶頸，這兩個部分可以做效能上改善。

第六章　結論與未來工作

XML 在大數據的應用下做為資料交換的格式有其重要性，而資料交換的時候如何確保資料的可信度以及真實度會是一個很重要的議題。

本研究提出了基於資料理解性的真實度模型 API，提出一個方式解決真實度不易表達也不易量化的問題。並且使用物件導向的觀念來設計模型，提高模型的程式重用以及程式設計彈性。並且實作範例模型放在 Apache Spark 叢集當中運行，以加速整體模型的處理效能。模型應用的串流 XML 文件，以解決串流 XML 資料因在串流當中結構的不確定性，而導致真實度驗證困難的問題。

本研究所提出的模型基於物件導向的設計理念，可以應用的不同的場域以及可以隨著資料特性的不同客製化出符合使用者資料特性的模型。這樣的特色可以有助於產學界降低模型的理解難度以及實作難度。

本研究提出真實度模型與模型 API，解決真實度不易表達也不易量化的問題，並且應用在串流 XML。後續研究可以朝向平行化計算以及串流資料處理優化或是 XML 結構對於效能的議題上來做後續探討。

參考文獻

[1] John Gantz and David Reinsel. Extracting value from chaos. *IDC iview*, 1142(2011): 1–12, 2011.

[2] The 5V's of big data. `https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data`. Retrieved on July, 2019.

[3] Nawsher Khan, Ibrar Yaqoob, Ibrahim Abaker Targio Hashem, Zakira Inayat, Mahmoud Ali, Waleed Kamaleldin, Muhammad Alam, Muhammad Shiraz, and Abdullah Gani. Big data: survey, technologies, opportunities, and challenges. *The Scientific World Journal*, 2014, 2014.

[4] Veracity: The most important "V" of big data. `https://www.gutcheckit.com/blog/veracity-big-data-v`. Retrieved on November, 2018.

[5] C. Lilley H. Thompson. XML Media Types. RFC 7303, RFC Editor, July 2014.

[6] Extensible Markup Language (XML) 1.0. `https://www.w3.org/TR/REC-xml`. Retrieved on December, 2018.

[7] W3C XML Schema Definition Language (XSD) 1.1. `https://www.w3.org/TR/xmlschema11-1`. Retrieved on December, 2018.

[8] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[9] Apache hadoop. `https://hadoop.apache.org`. Retrieved on December, 2018.

[10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. The hadoop distributed file system. In *MSST*, volume 10, pages 1–10, 2010.

[11] Apache spark. `https://spark.apache.org`. Retrieved on October, 2018.

[12] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In

Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, pages 2–2. USENIX Association, 2012.

[13] 陳宇威 (2017). 巨量 XML 真實程度模型之建構與介面設計. 碩士論文, 國立臺中科技大學, 台中市.

[14] Introduction to XML. `https://www.w3schools.com/xml/xml_whatis.asp`. Retrieved on July, 2019.

[15] 张丙奇, 白硕, and 赵章界. XML 数据相似度研究. *Computer Engineering*, 31(11), 2005.

[16] Hongjie Fan, Zhiyi Ma, Dianhui Wang, and Junfei Liu. Handling distributed XML queries over large XML data based on MapReduce framework. *Information Sciences*, 453:1–20, 2018.

[17] 曾偉誠 (2015). 雲端運算之 XML 巨量資料處理機制設計. 碩士論文, 國立臺中科技大學, 台中市.

[18] Philip Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1-3):217–239, 2005.

[19] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM (JACM)*, 26(3):422–433, 1979.

[20] Joe Tekli, Richard Chbeir, and Kokou Yetongnon. Semantic and structure based xml similarity: An integrated approach. In *COMAD*, pages 27–38. Citeseer, 2006.

[21] Joe Tekli and Richard Chbeir. A novel xml document structure comparison framework based-on sub-tree commonalities and label semantics. *Web Semantics: Science, Services and Agents on the World Wide Web*, 11:14–40, 2012.

[22] Joe Tekli, Richard Chbeir, Agma JM Traina, Caetano Traina Jr, and Renato Fileto. Approximate xml structure validation based on document–grammar tree similarity. *Information Sciences*, 295:258–302, 2015.

[23] Alsayed Algergawy, Richi Nayak, and Gunter Saake. Element similarity measures in xml schema matching. *Information Sciences*, 180(24):4975–4998, 2010.

[24] The data generation tool – xmlgen. https://projects.cwi.nl/xmark/downloads.html. Retrieved on January, 2019.

[25] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In *Proceedings of the eleventh international conference on data engineering*, pages 251–260. IEEE, 1995.

[26] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. Technical report, Stanford, 1997.

[27] A Standard Textual Interchange Format for the Object Exchange Model (OEM). http://infolab.stanford.edu/~mchughj/oemsyntax/oemsyntax.html. Retrieved on December, 2018.

[28] The Python Profilers. https://docs.python.org/3/library/profile.html. Retrieved on July, 2019.