Gestione dei dati Gis nel Progetto INNO

Deliverable A1

Versione	Data	Autori
0,2	02/10/15	Roberto Demontis Laura Muscas

1 Introduzione e obiettivi

Il WP1 del progetto INNO prevede la progettazione e realizzazione di un prodotto in grado di fornire servizi per il deployment, l'archiviazione, la gestione e l'interrogazione di dati Geografici e GIS tramite il meccanismo di tassellamento (tiling) vettoriale, con un approccio basato su un motore NoSQL, che garantisca scalabilità e replica.

L'utilizzo di un motore NoSQL implica che la struttura delle comuni basi di dati geografiche, basate su relazioni dovrà essere astratta e ridefinita quindi all'interno di database che non prevedono legami espliciti tra le informazioni (relazioni).

L'obiettivo è quello di implementare adeguati algoritmi che consentano l'accesso efficace a dati omogenei, in maniera analoga a quanto si ottiene con SQL in ambiente relazionale. Tali algoritmi, studiati in modo specifico per rispondere alle esigenze GIS, devono essere ottimizzati per rendere flessibile e fluida l'implementazione del front-end.

Il protocollo di comunicazione tra back-end e fron-tend deve essere progettato in modo da scambiare la minor quantità di informazione possibile: per privilegiare l'efficienza dell'infrastruttura verranno trasferiti solo dati numerici vettoriali, che saranno poi elaborati e trasformati in immagini dal front-end.

L'infrastruttura verrà calibrata in modo da poter essere testata anche per carichi di lavoro particolarmente intensi, in modo da evidenziare le doti di scalabilità e affidabilità poste come obiettivo tecnologico del progetto.

Verranno infine realizzati connettori e interfacce per l'accesso trasparente alla cloud da parte degli utilizzatori del front-end. Tali elementi permetteranno di interrogare da remoto la base dati che forniranno le informazioni necessarie alle applicazioni, attraverso semplici chiamate a funzioni API che implementano i protocolli dati e comunicazione; l'utente sviluppatore di applicazioni web potrà quindi utilizzare l'SDI senza essere a conoscenza della complessità interna del sistema di back-end.

All'interno del WP1 si possono individuare quattro obiettivi specifici:

1. individuazione del motore NoSQL più consono alla esigenze progettuali e alle risorse disponibili, come descritto nel paragrafo 4 e 5;
2. definizione dell'infrastruttura Hardware e Cloud, descritta nel paragrafo 2 e 3;
3. sviluppo e implementazione delle procedure di pre-trattamento dei dati GIS e del loro caricamento sul motore NoSQL, come descritto nel paragrafo 6;
4. realizzazione delle interfaccie di back-end per l'interrogazione dei dati ed il connettore al motore NoSQL, descritta nel paragrafo 7.

Nel paragrafo 8 vengono mostrate alcune statistiche realtive all'importazione di uno strato informativo nel sistema.

2 Descrizione dell'infrastruttura HD

L'infrastruttura Hardware è costituita da due server e dalle infrastrutture di rete necessarie a collegarli tra loro e verso internet. I server hanno ciascuno le seguenti caratteristiche:

Modello	Dell PowerEdge R420
Configurazione dello chassis	Chassis 3.5" Chassis con 4 Hot Plug Hard Drives
Numero Processori	2 Processori, ciascuno dei quali con caratteristiche (2.50GHz, 15M di cache, 7.2GT/s QPI, Turbo, 6C, 80W, DDR3-1600MHz)
Tipo e velocità della memoria	DIMM: 1600 MHz UDIMMs
Capacità di memoria	48GB
RAID	Raid Bios C8, RAID 1
Dischi Rigidi	2 dischi rigidi ciascuno dei quali con caratteristiche: 600GB, SAS 6Gbps, 2.5-in, 10K RPM Hybrid Hard Drive (Hot Plug) in 3.5-in Carrier
Adattatore di rete	Dual Port 1GBE

Tabella 1

I server hanno 12 core fisici con hyper threading, quindi vengono visti dal sistema operativo come 24 core.

3 Descrizione dell'infrastruttura Cloud Virtualizzata

La virtualizzazione sui server è stata realizzata tramite il software di virtualizzazione XenServer, un prodotto di Citrix basato su Xen. Le macchine virtuali definite sono 3 per ciascun server, tutte con sistema operativo ubuntu 12.04.

Delle 6 macchine virtuali risultanti 4 sono il back-end dell'infrastruttura, su queste quindi è installato il motore NoSQL. Come documentato nel paragrafo 5 si è scelta la versione community di couchbase versione 2.2. I nomi assegnati alle macchine sono innodata1.crs4.it, innodata2.crs4.it, innodata3.crs4.it, innodata4.crs4.it e costituiscono i nodi del cluster couchbase che gestisce i dati

Le restanti 2 macchine costituiscono il front-end del sistema, ossia il punto di accesso ai servizi che erogano i dati e agli applicativi per il loro caricamento e per i test. I software installati sono il web container Tomcat7, Apache2, PostgreSQL/PostGIS e le applicazioni web sviluppate nel progetto. I nomi assegnati a tali macchine sono inno.crs4.it e innotest.crs4.it.

Le caratteristiche di ciascuna delle 4 macchine virtuali contenenti il motore noSQL sono: 8

virtual core; 17538MB di RAM; in media 220GB di spazio disco per i dati. Le caratteristiche di ciascuna delle 2 macchine di servizio sono: 8 core virtuali; 8172MB di RAM e circa 80GB di spazio disco.

Come descritto ciascuna macchina ha a disposizione 8 core ma quelle dedicate a couchbase hanno una priorità più alta.

La macchina di frontend è connessa su 2 reti differenti: la prima fornisce connettività internet mentre la seconda viene usata per connettersi direttamente alla rete di backend su cui si trovano i nodi del cluster couchbase, non visibili all'esterno del CRS4 (Figura 1).

Allo stato attuale non è stato implementato un controllo sugli accessi alle porte usate da couchbase; in seguito si potrebbe prevedere di permettere un accesso diretto solo alle macchine del progetto INNO.

Al sistema è possibile aggiungere ulteriori macchine virtuali con istanze di nodi couchbasese previo collegamento alla rete di back-end ed inserimento dei nodi al cluster couchbase aumentandone capacità e performance. L'inserimento dei nodi può avvenire “a caldo” cioè senza interrompere il servizio.

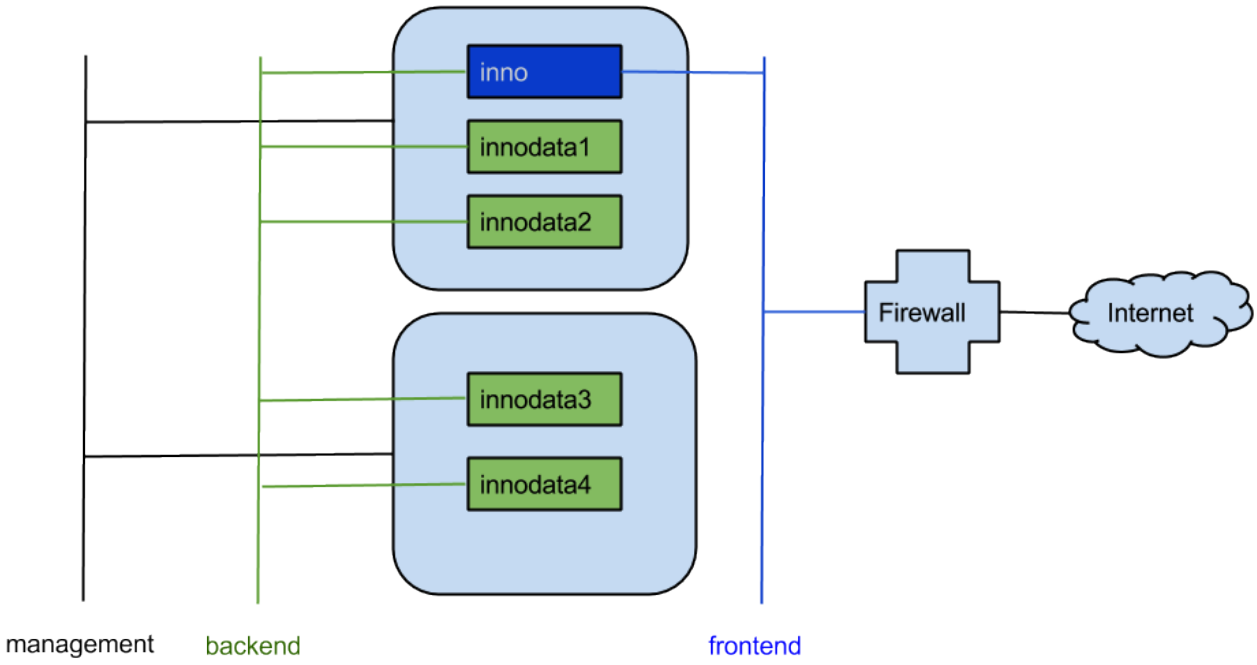


Figura 1

4 Stato dell'arte dei DB NoSQL

I database relazionali sono stati negli ultimi decenni il modello principale di gestione dei dati.

La disponibilità di un numero sempre crescente di applicazioni informatiche, soprattutto rivolte all'interazione tra utenti, ha incrementato notevolmente la generazione di contenuti da condividere e ha portato alla generazione di volumi di dati sempre maggiori. La disponibilità di moli di dati anche di ordine superiore ai petabyte ha incrementato le possibilità di analisi ma ha determinato problemi di gestione.

Con l'aumentare della quantità dei dati l'approccio strutturato dei database relazionali rallenta le sue prestazioni e il movimento NoSQL nasce proprio per cercare di ovviare al problema della gestione di grosse moli di dati eterogenei.

Un RDBMS è orientato alla rappresentazione strutturata dei dati, è un modello che ha dimostrato buone prestazioni in molte applicazioni ed è tutt'oggi il modello più utilizzato in ambito GIS.

Una interessante analisi effettuata da Solid IT[1] (Figura 2) evidenzia come ancora oggi le prime posizioni nella classifica di popolarità dei DBMS siano occupate da soluzioni basate sul modello relazionale. È necessario sottolineare che il punteggio viene calcolato in base all'interesse mostrato dall'utente ma è sicuramente indicativo di una forte preponderanza verso i sistemi SQL like.

256 systems in ranking, February 2015

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1439.72	+0.56
2.	2.	MySQL	Relational DBMS	1272.45	-5.06
3.	3.	Microsoft SQL Server	Relational DBMS	1177.48	-21.13
4.	↑ 5.	MongoDB	Document store	267.24	+16.35
5.	↓ 4.	PostgreSQL	Relational DBMS	262.34	+7.85
6.	6.	DB2	Relational DBMS	202.42	+2.29
7.	7.	Microsoft Access	Relational DBMS	140.54	+1.41
8.	8.	Cassandra	Wide column store	107.08	+8.34
9.	9.	SQLite	Relational DBMS	99.56	+3.37
10.	10.	Redis	Key-value store	99.21	+4.97

Figura 2

È comunque indubbio che il modello relazionale non può essere utilizzato efficacemente in ogni situazione e il problema si è manifestato in maniera particolare nella gestione dei Big Data; infatti lavorando con queste quantità di dati la complessità strutturale dei RDBMS rappresenta un problema e diventa necessario cercare soluzioni alternative.

Un interessante studio effettuato da Couchbase[2] su un campione di 1300 aziende intervistate, mette in evidenza le ragioni per cui si cercano soluzioni alternative al modello relazionale nella gestione di grosse moli di dati. Lo studio evidenzia che l'aspetto più critico sia la rigidità dello schema e la necessità di gestire molti dati eterogenei che i

1 http://db-engines.com/en/ranking
2 http://www.couchbase.com/press-releases/couchbase-survey-shows-accelerated-adoption-nosql-2012

database NoSQL, grazie alla loro scalabilità orizzontale, ossia alla possibilità di distribuire i dati e le operazioni su macchine fisiche differenti, possono meglio gestire.

È necessario comunque sottolineare che dal momento della loro nascita, ossia dal 2009, c'è stato un grandissimo movimento intorno ai database NoSQL, basti pensare che, secondo l'elenco consultabile su *http://nosql-database.org*, se ne contano 150 ed è complicato fare una valutazione sugli obiettivi che i database NoSQL hanno finora raggiunto.

I database NoSQL vengono utilizzati soprattutto da grandi compagnie web o imprese che gestiscono grossi siti web; per citare alcuni esempi Cassandra[3] è stato originariamente utilizzato all'interno di Facebook e oggi è usato da Twitter e Digg; Project Voldemort[4] è stato sviluppato e utilizzato da linkedin; servizi cloud come il database NoSQL di Amazon Simple e il servizio di memorizzazione e sincronizzazione cloud di Ubuntu One sono basati su CouchDB[5].

Per poter effettuare la scelta del sistema open source NoSQL da utilizzare nel progetto INNO sono stati esaminati i sistemi più in uso. Il primo passo è stato individuare quale tipologia di database NoSQL potesse essere più adatta al progetto.

I database NoSQL sono classificati in base al modello dei dati adottato, si distinguono quattro tipologie: Key-Values stores, Column-oriented database, Document database, Graph database.

Nel modello chiave-valore (Key-Values) il valore è un oggetto del tutto trasparente per il sistema, cioè non è possibile fare query sui valori ma solo sulle chiavi. Questi database risultano essere molto performanti per determinate applicazioni ma non consentono operazioni complesse di interrogazione e aggregazione; generalmente vengono utilizzati per conservare le informazioni relative alle sessioni di navigazione, ai profili utenti, ai carrelli dell'e-commerce.

I database orientati alle colonne (Column-oriented) organizzano i dati per colonne, le righe possono avere associate diverse colonne e tutte accessibili attraverso una chiave di riga. Questa tipologia di database sembra ricordare le tabelle relazionali ma c'è in realtà una differenza sostanziale: nei database orientati alle colonne le righe non hanno le stesse colonne e le colonne possono essere aggiunte alle righe in qualsiasi momento senza doverle aggiungere alle altre righe della stessa famiglia. La memorizzazione consecutiva di valori contenuti in ogni colonna permette, per query che coinvolgono pochi attributi selezionati su un gran numero di record, di ottenere tempi brevi di risposta; inoltre si possono applicare facilmente tecniche di compressione dei dati. Questi database sono pensati per il mantenimento dei contatori, per le piattaforme CMS e per sostenere le pesanti sessioni di scrittura di importanti volumi di dati come nel caso delle operazioni di aggregazione dei log.

I database orientati ai documenti (Document database) sono simili a tabelle hash con un unico campo di identificazione e valori che possono essere di qualunque tipo. I documenti (XML, JSON, BSON, etc.) possono contenere strutture nidificate. A differenza dei database basati su chiave-valore i sistemi orientati ai documenti supportano indici secondari, replicazione e interrogazioni ad hoc. Questi database sono utili per le piattaforme di content management, per le piattaforme di blogging, per le Web analytics, le analisi real-time e le applicazioni di e-commerce.

I database a grafo (Graph database) rappresentano un caso particolare di database orientati ai documenti in cui i documenti rappresentano sia i nodi che le relazioni che

3 https://cassandra.apache.org/
4 http://www.project-voldemort.com/voldemort/
5 http://couchdb.apache.org/

interconnettono i nodi. Questi database sono utili per risolvere i problemi di memorizzazione e gestione di dati interconnessi come le informazioni dei social network, le informazioni di tracciablità delle merci e le transazioni monetarie.

5 Il motore db NoSQL scelto e installato su Cloud

Per poter utilizzare un motore NoSQL la struttura delle comuni basi di dati geografiche, basate su relazioni, deve essere ridefinita all'interno di database che non prevedeno legami espliciti tra le informazioni (relazioni).

Tale approccio non può però essere integrale, in quanto delegare completamente su client la risoluzione delle relazioni sui dati, soprattutto quelle spaziali, diminuisce notevolmente la performance per i tempi di elaborazione necessari e le dimensioni del dato da trasmettere.

Per risolvere il problema i motori NoSQL più popolari insieme all'operazione di indicizzazione "Map", che ricava un valore data una chiave, utilizzano anche l'operazione "Reduce" che implementa una parziale elaborazione sui risultati prima della restituzione. Più l'operazione di "Reduce" è complessa più la performance del motore NoSQL diminuisce.

Nel progetto INNO i meccanismi di restituzione del dato non prevedono operazioni di "Reduce" ma un pretrattamento del dato per generare valori più attinenti all'uso che se ne vuole fare. Tale valore è in realtà un documento JSON che rappresenta un tassello contenente le geometrie semplificate degli elementi di uno strato informativo, indicizzato tramite una chiave generata usando le coordinate che la localizzano ed il livello di zoom unito all'identificativo univoco dello strato informativo.

La parte alfanumerica degli elementi dello strato informativo viene invece memorizzata in un documento JSON separato, gestito in qualità di documento e acceduto tramite chiave o tramite viste. Tale scelta è dettata principalmente dalla necessità di limitare il numero di chiavi gestite dal sistema.

Avendo esaminato le caratteristiche delle diverse tipologie di database NoSQL ed avendo esculso elaborazioni su back-end per la creazione del tassello con le geometrie si è scelto di utilizzare, per il progetto INNO, un database orientato ai documenti. In particolare sono stati presi in esame i motori MongoDB[6] e Couchbase[7].

Sia MongoDB che Couchbase sono risultati robusti, efficienti e agevolmente gestibili ed installabili in ambiente cloud. Le differenze sono emerse nella gestione del dato spaziale dove per l'indicizzazione delle geometrie poligonali non sono presenti valide soluzioni in MongoDB che usa chiavi di tipo geohash non utili per l'indicizzazione di aree.

La gestione di indici spaziali in MongoDB per geometrie poligonali è infatti attualmente oggetto di studio[8] ma non è ancora disponibile un modulo software. Di conseguenza è stato scelto Couchbase quale motore NoSQL che metta a disposizione indici spaziali basati su R-Tree. Inoltre il motore ha una buona documentazione, un'interfaccia di amministrazione evoluta e disponibilità di software di sviluppo (SDK) per diversi linguaggi di programmazione.

6 http://www.mongodb.org/
7 http://www.couchbase.com/
8 http://www.slideshare.net/nknize/rtree-spatial-indexing-with-mongodb-mongodc

6 Preparazione e tassellamento dei dati GIS

Si è previsto di utilizzare come formato per il dato GIS in ingresso lo shapefile, soddisfacente alcune caratteristiche di base (Tabella 2). Se il dato in origine non è uno shapefile l'utilizzatore del sistema dovrà convertirlo in quel formato; tale operazione è possibile nella quasi totalità dei sistemi che elaborano dati GIS.

Prima di essere caricato sul database NoSQL il dato viene elaborato e trasformato in documenti JSON. Le proprietà alfanumeriche e le geometrie degli elementi sono trattate separatamente. Le geometrie generano il tassellamento dello strato dove in un tassello possono risiedere diversi elementi dello stesso strato. Le proprietà alfanumeriche invece risiedono in un documento riferito all'elemento a cui appartengono.

Le caratteristiche che lo strato informativo deve avere non sono pienamente vincolanti, alcune potranno essere modificate opportunamente previa modifica delle procedure di pre-trattamento del dato (e.g il sistema di riferimento dei dati).

Nomi delle proprietà	Alcune limitazioni sul nome delle proprietà
Tipologia geometrie	I tipi permessi sono: linea, multilinea, poligono, multipoligono, punto, multipunto.
Unicità del tipo	Gli elementi devono essere tutti dello stesso tipo
Validità geometrie	Le geometrie devono essere valide (o semplici) secondo le specifiche openGIS[9]
Unicità dell'identificativo dello strato informativo	Il nome dello shapefile sarà l'identificativo univoco dello strato informativo.
Sistema di riferimento	Il sistema di riferimento delle geometrie deve essere il WGS84
Intersezioni	Non è consigliato inserire strati informativi di tipo poligonale che contengono intersezioni tra elementi diversi

Tabella 2

Il sistema utilizza lo standard "Slippy map tilenames"[10] per la definizione dei tasselli con dimensioni di 256 pixel in larghezza ed altezza.

L'utilizzo del tassellamento del dato GIS comporta due specifici problemi relativi al dimensionamento ed al numero dei tasselli. Tali problemi sono comunque direttamente collegati alle risorse disponibili in termini di velocità di trasferimento dei dati e quantità di memoria disponibile.

Il dato che esalta di più le caratteristiche del sistema è uno strato informativo contenente geometrie poligonali ciascuna delle quali composta da un elevato numero di vertici (nell'ordine del milione). Viceversa il dato che ne evidenzia i limiti è uno strato con un elevato numero di geometrie poligonali ciascuna con estensione dell'ordine di alcuni gradi e pochissimi vertici.

9 http://portal.opengeospatial.org/files/?artifact_id=25355
10 http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames

I tasselli devono essere, per specifiche di progetto, leggeri (<= 20 kb) quindi, per contenere il massimo dell'informazione, le coordinate dei vertici sono proiettate relativamente all'origine del tassello e rappresentate in formato esadecimale.

Per evitare la perdita d'informazione soprattutto ai livelli di zoom bassi è presente un meccanismo di paginazione. Nell'inserimento degli elementi in un tassello viene usato un valore di priorità che, nel caso dei poligoni, è l'area coperta, per le linee è invece la lunghezza. Il caso dei punti è più complesso.

Per ottenere una copertura omogenea nel tassello vengono calcolati: la chiave geohash di 7 cifre del punto (cioè l'identificativo del buffer di 70 metri che lo contiene) ed il numero dei punti con stessa chiave geohash. Il primo dei punti nella tabella, con una particolare chiave, assume come priorità il numero di elementi “simili”, cioè con la stessa chiave geohash. Viceversa i punti “simili” assumono come priorità lo stesso valore ma in negativo.

Per limitare il numero dei tasselli, nel caso di strati informativi con geometrie poligonali estese, il sistema genera i macro-tasselli. Questi rappresentano un insieme di tasselli adiacenti interamente contenuti in una delle geometrie dello strato ed hanno sempre dimensione rettangolare . Un *macro tassello* è identificato da un tassello “origine”, il livello di zoom e le estensioni, in numero di tasselli, rispetto alla x e alla y. L'uso dei macro-tasselli permette un risparmio elevato in numero di chiavi e documenti da generare nel caso di geometrie poligonali estese (un esempio nel paragrafo 8).

Il procedimento di pre-trattamento segue alcuni semplici passi ed è implementato da alcune funzioni plpsql in ambiente PostgreSQL – PostGIS. Tale scelta è dettata dalla necessità di utilizzare un sistema con un vasto repertorio di funzioni e robusto, cioè senza problemi di crash di sistema o di utilizzo di memoria in fase di elaborazione di grandi quantità di dati.

Il procedimento segue i seguenti passi:

1. import del file nel DBMS PostgreSQL
2. procedura di semplificazione e preparazione
3. creazione dei documenti JSON di tipo 'tile'
4. creazione dei documenti JSON di tipo 'info' e 'layer'
5. scrittura su disco dei documenti

Successivamente i documenti vengono compressi in formato .zip e caricati su couchbase tramite l'applicativo cbdocloader presente nella libreria di Couchbase.

Il sistema sarà corredato da una applicazione che permetterà l'accesso alla cloud da parte degli utilizzatori del back-end per le operazioni di inserimento degli strati informativi. Tale operazione sarà eseguita in maniera totalmente trasparente rispetto alla complessità della procedura sopra descritta. L'utilizzatore potrà decidere quali livelli di zoom abilitare in base alla precisione del dato che intende importare nel sistema.

7 Back-end, interfacce e connettori per il motore noSQL

Le interfacce ed il connettore al motore NoSQL Couchbase sono implementate da una applicazione web operante sulla macchina virtuale all'indirizzo http://inno.crs4.it/inno.

L'applicazione è installabile su altre macchine senza doverne modificare il codice così da aggiungere ulteriori punti di accesso ai dati del cluster couchbase.

I servizi esposti dall'applicazione permettono un accesso trasparente alla cloud da parte degli utilizzatori del front-end. Tali metodi consentiranno l'interrogazione da remoto della base dati e forniranno le informazioni necessarie alle applicazioni attraverso semplici chiamate a funzioni API che implementano i protocolli dati e comunicazione; l'utente sviluppatore di applicazioni web potrà quindi utilizzare il sistema senza essere a conoscenza della complessità interna del sistema di back-end.

L'interfaccia implementata espone i seguenti servizi diferenziati logicamente per categoria:

Categoria	Metodo	Descrizione
Catalogo	GetLayerList()	Restituisce il documento JSON con la lista degli strati informativi nel cluster couchbase.
Catalogo	GetLayerInfo()	Restituisce il documento JSON dello strato informativo con chiave specificata come parametro.
Tiling	GetTile()	Restituisce il documento JSON con le geometrie degli elementi che ricadono nel tassello indicato come chiave.
Query	getValue()	Restituisce il documento JSON con i soli valori di una proprietà degli elementi nel tassello indicato come chiave.
Query	getFeature()	Restituisce il documento JSON con le informazioni dell'elemento con identificatore il valore indicato come chiave.

Tabella 3

Attualmente non sono implementate tutte le operazione CRUD, cioè oltre alla lettura non è permessa la creazione, modifica e cancellazione dei singoli elementi di uno strato informativo. Tali operazioni sono però facilmente implementabili sui dati alfanumerici, possibili ma più complicate per le parti geometriche.

Nell'Header HTTP della response verrà codificato l'esito dell'operazione e eventuali messaggi di errore.

Di seguito la descrizione dei documenti restituiti dalle chiamate ai servizi:

GetLayerList(): Lista degli strati informativi nel sistema

Chiamata: http://inno.crs4.it/inno/json/list

Parametri: nessuno

Il documento restituito è un array di oggetti del tipo descritto in Tabella 4

GetLayerInfo(strato:testo)

Chiamata: http://inno.crs4.it:8080/inno/json/layer/*<strato>*

Parametri: *strato* è l'identificativo univoco nel sistema dello strato informativo

Il documento restituito contiene le informazioni dello strato. I campi del documento sono descritti in Tabella 4

Nome	Tipo	Descrizione
innoname	Testo	Identificativo univoco dello strato informativo
bbox	GeoJSon	Il bounding Box in Wgs84 che contiene l'intero strato informativo
vertices	Integer	Il numero totale di vertici nello strato informativo
count	Integer	Il numero di elementi nello strato informativo
type	Testo	Il tipo di geometrie contenuto in tutti gli elementi. Può essere: “MULTIPOINT”, “POINT”, “MULTILINESTRING”, “LINESTRING”, “MULTIPOLYGON”, “POLYGON”
attributes	Array<Attribute>	Le caratteristiche delle proprietà alfanumeriche dello strato informativo

Tabella 4

Le caratteristiche delle proprietà alfanumeriche degli elementi dello strato informativo sono rappresentate da un oggetto Attribute con i seguenti campi:

Nome	Tipo	Descrizione
name	Testo	Il nome della proprietà così come nello shapefile
type	Testo	Il tipo della proprietà

Tabella 5

Esempio:

```
{
     "_innoname_":"comuni",
     "_bbox_":{"type":"Polygon",
                    "coordinates":  [[[8.130769799999999,38.859124],
                                     [8.130769799999999,41.3132405],
                                     [9.8270429,41.3132405],[9.8270429,38.859124],
                                     [8.130769799999999,38.859124]]]},
                     "vertices":314610,
                     "count":377,
                     "type":"MULTIPOLYGON",
                     "attributes":[
                           {"name":"gid","type":"integer"},
                      ... ]}
}
```

GetFeature(strato:testo,feature:testo)

Chiamata: http://inno.crs4.it:8080/inno/json/feature/*<strato>/<feature>*

Parametri: *strato* è l'identificativo univoco nel sistema dello strato informativo; *feature* è l'identificativo univoco nel sistema dell'elemento.

Il documento contiene le proprietà alfanumeriche di un elemento e l'estenzione come descritto in Tabella 6.

Nome	Tipo	Descrizione
gid	integer	Identificatore numerico nel dato di origine dell'elemento
id	Testo	Identificatore dell'elemento nel sistema
<layername>:bbox	GeoJSON	Il bounding Box in Wgs84 che contiene l'elemento
<attribute_1>	< type attribute_1>	Nome e valore di una proprietà dell'elemento
....	...	...
<attribute_n>	< type attribute_n>	Nome e valore di una proprietà dell'elemento

Tabella 6

Esempio:


```
{    "_comunibbox_": { "type": "Polygon",
                       "coordinates": [[[8.13077,40.65175],[8.13077,40.8693],
                                  [8.6436,40.8693],[8.6436,40.65175],[ 8.13077,40.65175 ] ] ] },
     "gid": 128, "istat": 90064, "nome": "Sassari",
     "regione": "Sardegna","subregione": "Sassarese", "id": "comuni:001"
}
```


GetValue(strato:testo,attributo:testo,x:integer,y:integer,zoom:integer)

Chiamata: http://inno.crs4.it:8080/inno/json/value/*<strato>/<attributo><x>/<y>/<zoom>*

Parametri: *strato* è l'identificativo univoco nel sistema dello strato informativo; attributo è il nome della proprietà; *x,y* e *zoom* identificano un tassello.

Il documento restituito contiene un array di oggetti, descritti in Tabella 7, con l'identificativo dell'elemento ed il valore della proprietà per tutti gli elementi nel tassello alle coordinate x, y e livello zoom

Nome	Tipo	Descrizione
id	Testo	Id dell'elemento nello strato
v	Testo	Valore dell'attributo

Tabella 7

Esempio


```
{ "subregione": [{"id":"comuni:158","v":"Sulcis-Iglesiente"},
                 {"id":"comuni:249","v":"Sulcis-Iglesiente"},
                 {"id":"comuni:112","v":"Sulcis-Iglesiente"}]
}
```

getTile(strato:testo,x:integer,y:integer,zoom:integer)

Chiamata: http://inno.crs4.it:8080/inno/json/value/<strato>/*<x>*/*<y>*/*<zoom>*

Parametri: *strato* è l'identificativo univoco nel sistema dello strato informativo; *x*, *y* e *zoom* identificano un tassello.

I documenti contengono le geometrie codificate di tutti gli elementi nel tassello identificato da x, y e zoom

Nome	Tipo	Descrizione
id	Testo	Identificatore univoco del tassello <layername>:<x>:<y>:<zoom>(:<page>) page è aggiunto solo se page è > 1
bbox	GeoJSON	Bounding box del tassello
page	integer	Numero di pagina rispetto all'insieme delle pagine del tassello
pages	integer	Numero pagine con i dati delle geometrie nel tassello
objs	Array[Geometria]	Contiene gli elementi o dati del layer nel tassello

Tabella 8

Gli elementi geometrici sono rappresentati da un oggetto Geometria con I seguenti campi:

Nome	Tipo	Descrizione
id	Testo	Identificatore dell'elemento all'interno dello strato (l'identificativo nel sistema è ottenuto aggiungendo come prefisso l'identificatore dello strato più ':'
g	Testo	La rappresentazione della porzione nel tassello della geometria dell'elemento, opportunamente codificata

Tabella 9

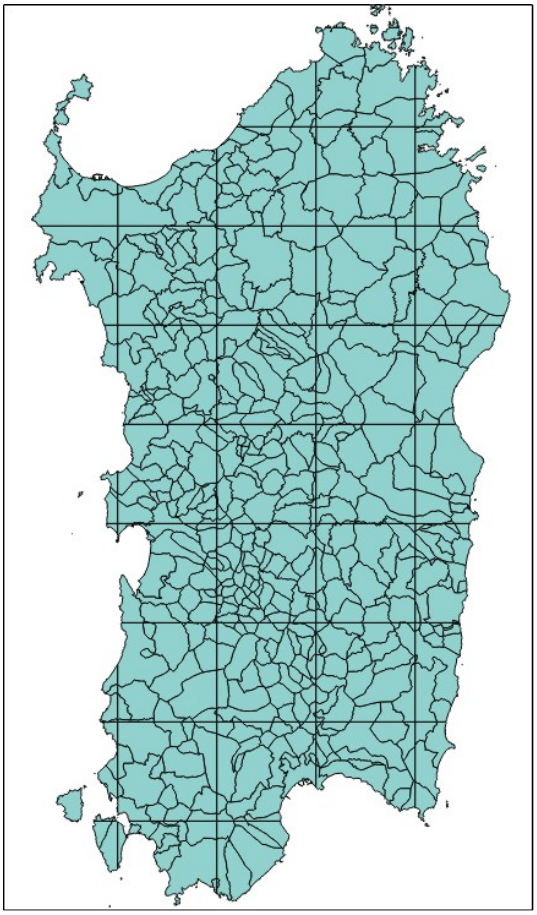
Esempio


```
{
    "bbox":{"type":"Polygon",
            "coordinates":[[[9.49219,38.82259],[9.49219,39.09596],[9.84375,39.09596],
                           [9.84375,38.82259],[9.49219,38.82259]]]},
    "objs": [ {"id":"143","g":"3((17001700160016001500 1700))"},
                {"id":"483","g":"3((230b230c220c230c220c220b230b))"},
                {"id":"488","g":"3((210d200d200c200d210d))"},
                {"id":"495","g":"3((240b230b230a240a240b))"},
                {"id":"550","g":"3((20092109210a200a2009))"}],
    "id":"comuni:539:391:10",
    "page":1,
    "pages":1
}
```

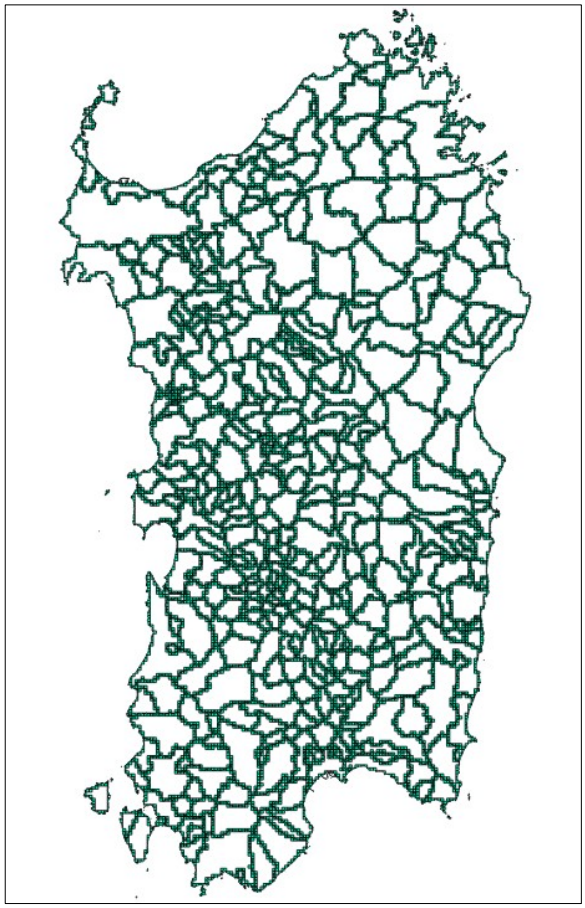
8 Un Esempio: il layer dei comuni della sardegna

Di seguito le statistiche realtive all'inserimento di uno strato informativo, relativamente leggero, dei poligoni relativi ai comuni della sardegna.

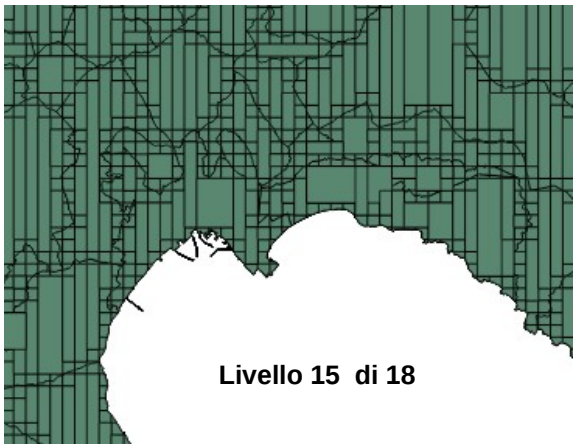
La tabella 10 mostra alcuni risultati del caricamento dello strato nel sistema e le immaggini successive rappresentano graficamente i tasselli evidenziando il numero dei documenti prodotti.



Livello 10



Livello 15 di 18 *(gli spazi bianchi interni ai poligoni contengono i macro tasselli)*



Livello 15 di 18

Nome	Descrizione
Descrizione	poligoni relativi ai comuni della sardegna fonte ISTAT
Identificativo nel sistema	comuni
Formato origine	Shapefile geometrie in WGS84
Dimensione	5.3 MB
Tipo geometrie	Multipolygon
Elementi	377
Vertici	314,61
N° di tasselli senza macro tasselli (livello 18)	1.639.535
N° di tasselli con macro tasselli (livello 18)	128795
Tempo complessivo per l'elaborazione e l'inserimento	2h

Tabella 10