Contents

MapProxy e Sistemi di Riferimento Geodetici

Fondamenti Teorici per la Configurazione del Servizio Catastale Italiano

Autore: Antonio Rocca - GeoAnalyst s.r.l.
Data: Dicembre 2025
Livello: Avanzato - Teoria Geodetica e Architettura Software

Indice

---

1. Introduzione

Contesto del Problema

Il servizio WMS dell'Agenzia delle Entrate rappresenta un caso di studio paradigmatico nell'interoperabilit dei dati geospaziali. La cartografia catastale italiana, formata storicamente su oltre 800 diversi sistemi di coordinate locali, stata uniformata nel sistema RDN2008 (EPSG:6706), mentre la maggior parte delle applicazioni web moderne opera in Web Mercator (EPSG:3857) e i software topografici professionali utilizzano proiezioni UTM.

Questa situazione genera una **triplice sfida**:

1. **Geodetica:** Conversione accurata tra datum e proiezioni diverse

2. **Prestazionale:** Servizio WMS lento (2-5 secondi per tile)
3. **Interoperabilit:** Formati non standard per applicazioni web

MapProxy risolve queste sfide fungendo da **gateway intelligente** che: - Acquisisce dati nel sistema nativo (EPSG:6706) - Li memorizza in cache - Li serve in formati standard riproiettati on-demand

Obiettivi Didattici

Questo documento esplora: - I principi geodetici alla base delle trasformazioni coordinate - L'architettura software di MapProxy - Le scelte progettuali per ottimizzare precisione e prestazioni - Le implicazioni pratiche dei diversi sistemi di riferimento

---

2. Fondamenti Geodetici

2.1 Il Problema della Rappresentazione Terrestre

La Terra approssimativamente uno **sferoide oblato** (ellissoide di rotazione schiacciato ai poli). Rappresentare una superficie curva su un piano bidimensionale richiede una **proiezione cartografica**, che inevitabilmente introduce distorsioni in: - Angoli (conformit) - Aree (equivalenza) - Distanze (equidistanza) - Forme

Teorema di Gauss: Non esiste proiezione che preservi simultaneamente angoli, aree e distanze.

2.2 Datum Geodetico

Un **datum** definisce: 1. **Ellissoide di riferimento:** Modello matematico della Terra 2. **Punto di origine:** Coordinate del punto fondamentale 3. **Orientamento:** Direzione degli assi coordinati

Esempio storico italiano: - **Monte Mario (Roma 1940):** Datum locale, ellissoide Internazionale 1924 - **WGS84:** Datum globale, ellissoide WGS84 - **ETRS89/ETRF2000:** Datum europeo, "congelato" rispetto alla placca eurasiatica

2.3 Trasformazioni di Datum

Convertire coordinate tra datum diversi richiede una **trasformazione tridimensionale** che include:

Trasformazione di Helmert a 7 parametri:

```
X_new = t_x + (1 + s) * R * X_old
```

Dove: - `t_x, t_y, t_z` = Traslazioni (metri) - `s` = Fattore di scala (ppm) - `R` = Matrice di rotazione (3 angoli in secondi d'arco)

Per Italia: Monte Mario -> ETRS89: - DeltaX circa -104 m - DeltaY circa -49 m
- DeltaZ circa +10 m - Rotazioni circa 0.5" (circa 15 metri sulla superficie)

2.4 Proiezioni Cartografiche

Proiezione Cilindrica (UTM, Web Mercator) Formula di Mercatore (semplificata):

```
x = lambda - lambda0
y = ln(tan(pi/4 + phi/2))
```

Dove: - `phi` = latitudine - `lambda` = longitudine - `lambda0` = meridiano centrale

Caratteristiche: - **Conforme:** Preserva angoli localmente - **Non equivalente:** Distorsione areale cresce con latitudine - **Limite:** phi < +o-85.06deg (Web Mercator)

Proiezione UTM (Universal Transverse Mercator) Variante della proiezione di Mercatore **traversa** (cilindro tangente a un meridiano):

Parametri UTM: - Fusi: 60 zone da 6deg di longitudine - Fattore di scala al meridiano centrale: k0 = 0.9996 - False Est: 500,000 m - False Nord: 0 m (emisfero nord)

Distorsione lineare UTM:

```
k = k0 * (1 + x^2/(2*R^2*k0^2))
```

Dove: - `x` = distanza dal meridiano centrale - `R` = raggio terrestre medio (~6371 km)

Per x = 180 km (bordo fuso): - Distorsione circa 0.04% (40 cm/km)

Italia coperta da: - Fuso 32N: 6degE - 12degE (EPSG:32632) - Fuso 33N: 12degE - 18degE (EPSG:32633)

2.5 Accuratezza Posizionale

La **cartografia catastale italiana** ha accuratezza variabile:

Tipo Mappa	Scala Originale	Metodo Rilievo	Accuratezza sigma
Catasto nuovo impianto	1:2000	Tacheometria	+o-2 m
Catasto ex Urbano	1:1000-500	Trilaterazione	+o-0.5-1 m
Catasti preunitari	Varie	Squadro e catena	+o-5-10 m
Aggiornamenti GNSS	-	GPS/GNSS RTK	+o-0.05 m

Implicazione pratica: L'accuratezza della riproiezione (+o-0.1 m) **trascurabile** rispetto all'accuratezza intrinseca dei dati.

---

3. Sistemi di Riferimento Italiani

3.1 Evoluzione Storica

3.1.1 Catasti Preunitari (pre-1886)

- Sistemi locali eterogenei
- Datum: Punti trigonometrici regionali
- Unit: Trabucchi, canne, passi
- Orientamento: Magnetico o astronomico locale

3.1.2 Nuovo Catasto Terreni (1886-1940)

- **Datum:** Bessel 1841 (orientamento a Roma-Quirinale)
- **Proiezione:** Cassini-Soldner locale per foglio
- **Scala:** 1:2000 prevalente
- **Copertura:** Progressiva, completata ~1950

3.1.3 Roma 1940 - Monte Mario (1940-2008) Unificazione nazionale: - **Ellissoide:** Internazionale 1924 (Hayford) - **Datum:** Monte Mario (Roma, phi=41deg55'25.51", lambda=12deg27'08.40") - **Proiezioni:** - Gauss-Boaga (2 fusi: Ovest/Est) - UTM (fusi 32-34)

Coordinate Gauss-Boaga: - EPSG:3003 (Fuso Ovest): origine 9degE - EPSG:3004 (Fuso Est): origine 15degE

3.1.4 ETRS89 Realizzazioni Italiane (1991-2008)

- **ETRF89:** Prima realizzazione europea
- **ETRF2000 (epoca 2008.0):** Adottata ufficialmente in Italia

Decreto 10 novembre 2011: > "Il Sistema di riferimento geodetico nazionale la realizzazione ETRF2000 - all'epoca 2008.0 - del Sistema di riferimento terrestre europeo ETRS89."

3.1.5 RDN2008 - Rete Dinamica Nazionale (2008-presente)

- **Codice EPSG:** 6706
- **Datum:** ETRF2000(2008.0)
- **Ellissoide:** GRS80 (identico a WGS84 entro 0.1 mm)
- **Coordinate:** Geografiche (phi, lambda, h)
- **Rete fiduciale:** 99 stazioni permanenti GNSS

Parametri ellissoide GRS80:

```
a (semiasse maggiore) = 6,378,137 m
f (schiacciamento) = 1/298.257222101
b (semiasse minore) = 6,356,752.314 m
```

3.2 Confronto RDN2008 vs WGS84

Differenze pratiche:

Parametro	WGS84	GRS80/ETRS89	Differenza
Semiasse maggiore	6378137 m	6378137 m	0
Schiacciamento	1/298.257223563	1/298.257222101	~0.1 mm
Deriva placca	~2 cm/anno	0 (fisso 1989)	Cumulativa

Per Italia (2025): - Deriva WGS84->ETRS89: ~70 cm Nord-Est - **Rilevante per:** Navigazione GNSS ad alta precisione - **Trascurabile per:** Cartografia catastale (sigma > 1 m)

3.3 UTM per l'Italia

Fuso 32N (EPSG:32632): - Meridiano centrale: 9degE - Copertura: 3degE - 15degE - Regioni: Piemonte, Liguria, Valle d'Aosta, Lombardia occidentale, Toscana occidentale - Distorsione massima: ~0.04% ai bordi

Fuso 33N (EPSG:32633): - Meridiano centrale: 15degE - Copertura: 9degE - 21degE - Regioni: Lazio, Abruzzo, Molise, Campania, Basilicata, Calabria, Puglia, Sicilia - Include Casalattico (13.75degE) con distorsione <0.02%

Scelta del fuso:

```
Se lambda < 12degE -> Usa fuso 32N
Se lambda >= 12degE -> Usa fuso 33N
```

Zona di sovrapposizione (9deg-15degE): Entrambi i fusi sono definiti, ma per minimizzare distorsione: - Usa 32N se lambda < 12deg - Usa 33N se lambda >= 12deg

3.4 Web Mercator (EPSG:3857)

Storia: - Introdotto da Google Maps (2005) - Adottato universalmente per web mapping - **Non conforme EPSG** fino al 2008 (poi assegnato codice 3857)

Caratteristiche distintive:

1. **Proiezione:** Mercatore cilindrica diretta

2. **Ellissoide:** Sfera (R = 6378137 m) invece di WGS84 ellissoide
3. **Limite latitudine:** +o-85.051129deg (per avere mappa quadrata)

Formula coordinate:

```
x = R * lambda
y = R * ln(tan(pi/4 + phi/2))
```

Conversione da lat/lon:

```python
import math

R = 6378137.0  # Raggio terra (metri)

def latlon_to_webmercator(lat, lon):
    x = R * math.radians(lon)
    y = R * math.log(math.tan(math.pi/4 + math.radians(lat)/2))
    return x, y

# Esempio: Roma (41.9degN, 12.5degE)
x, y = latlon_to_webmercator(41.9, 12.5)
# x circa 1,391,500 m
# y circa 5,149,000 m
```

Distorsione areale Web Mercator:

La scala varia con la latitudine:

```
k = 1 / cos(phi)
```

Latitudine	Scala k	Distorsione
0deg (Equatore)	1.00	0%
40deg (Centro Italia)	1.31	+31%
45deg (Nord Italia)	1.41	+41%
60deg	2.00	+100%
85deg (Limite)	11.5	+1050%

Implicazioni: - **Angoli:** Preservati localmente (conforme) - **Aree:** Fortemente distorte (Italia +35% rispetto a realt) - **Distanze:** Utilizzabili solo per calcoli approssimativi

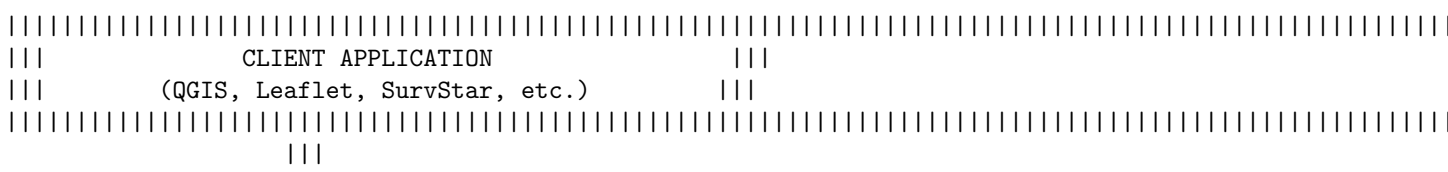
Perch usato per web? 1. **Semplicit matematica:** Calcoli veloci 2. **Tile quadrate:** Mappa mondiale 256x256 pixel a zoom 0 3. **Continuit:** Nessuna discontinuit ai bordi fuso (come UTM) 4. **Standard de facto:** Interoperabilit universale

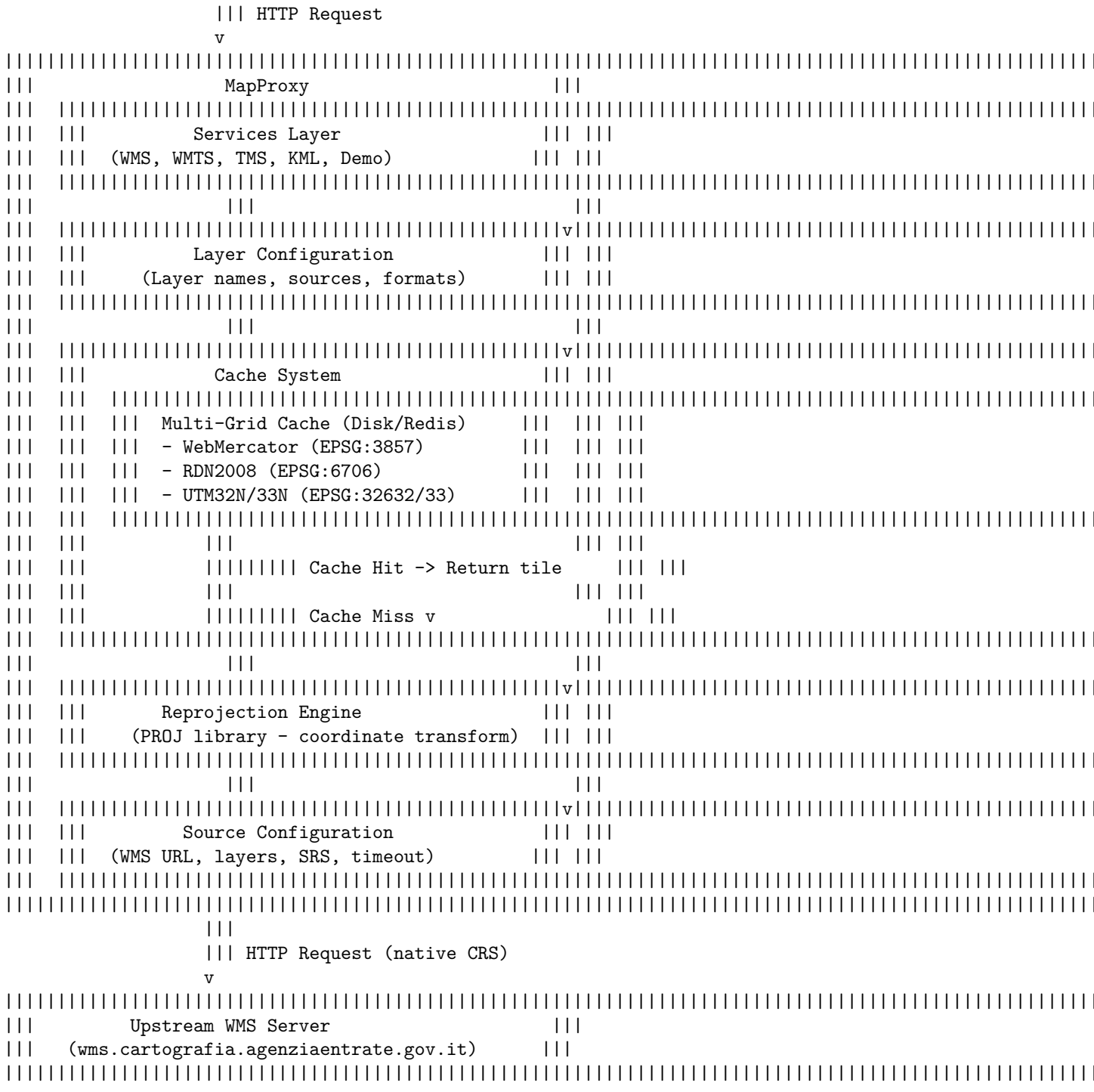
---

4. Architettura MapProxy

4.1 Panoramica Architetturale

MapProxy un **tile caching proxy** modulare scritto in Python che implementa lo standard **OGC WMS/WMTS/TMS**.





4.2 Componenti Principali

4.2.1 Services Layer Implementa protocolli standard:

WMS (Web Map Service): - GetCapabilities: Metadata servizio - GetMap: Richiesta immagine raster - GetFeatureInfo: Query attributi (opzionale)

WMTS (Web Map Tile Service): - RESTful: `/wmts/layer/grid/zoom/col/row.format` - KVP: Query string con parametri tile

TMS (Tile Map Service): - URL: `/tiles/layer/grid/zoom/col/row.format` - Schema: Slippy Map (standard Google/OSM)

4.2.2 Grid System Un **grid** definisce la struttura di tiling:

```yaml
grids:
  webmercator:
    srs: EPSG:3857
    origin: nw                    # Nord-Ovest
    bbox: [-20037508.34, -20037508.34, 20037508.34, 20037508.34]
    res: [156543.03392804097,     # Zoom 0: 156 km/pixel
          78271.516964020484,     # Zoom 1: 78 km/pixel
          39135.758482010242,     # Zoom 2: 39 km/pixel
          # ... fino a zoom 18
          0.5971642834779395]     # Zoom 18: 0.6 m/pixel
```

Calcolo risoluzione:

```
res[z] = C / (256 * 2^z)
```

Dove: - `C` = Circonferenza terrestre Web Mercator = 2piR circa 40075017 m - `z` = Livello zoom

Tile addressing:

```
tile_x = floor((x - bbox_minx) / (res * tile_width))
tile_y = floor((bbox_maxy - y) / (res * tile_height))
```

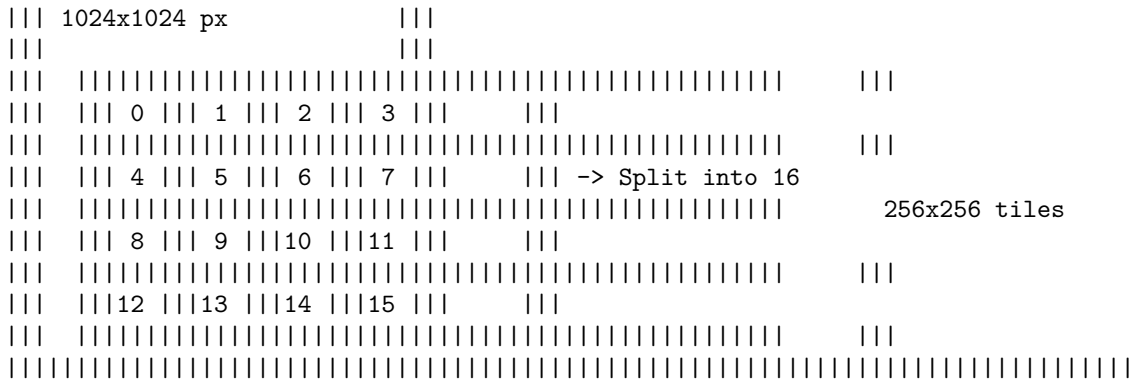
4.2.3 Cache System **Cache hit flow:**

```python
def get_tile(layer, grid, z, x, y):
    cache_key = f"{layer}/{grid}/{z}/{x}/{y}"

    # 1. Check cache
    tile = cache.get(cache_key)
    if tile:
        return tile

    # 2. Cache miss - acquire lock
    with tile_lock(cache_key):
        # 3. Check again (another thread might have filled)
        tile = cache.get(cache_key)
        if tile:
            return tile

        # 4. Generate tile
        tile = generate_tile(layer, grid, z, x, y)

        # 5. Store in cache
        cache.set(cache_key, tile)

    return tile
```

Meta-tiling:

Invece di richiedere 1 tile per volta al WMS upstream, MapProxy richiede una **meta-tile** (es. 4x4 tiles) e la divide:

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||| WMS Request (large)     |||
```



Vantaggi: - Riduce numero richieste WMS (da 16 a 1) - Migliora efficienza upstream - Riduce overhead HTTP

Configurazione:

```
meta_size: [4, 4]       # Richiede 4x4 = 16 tiles per volta
meta_buffer: 20         # Pixel extra per evitare artefatti bordi
```

4.2.4 Reprojection Engine MapProxy usa **PROJ** (Cartographic Projections Library) per trasformazioni coordinate.

Pipeline trasformazione:

```
Source CRS (EPSG:6706)
    v
1. Geodetic -> Geocentric (X,Y,Z)
    v
2. Helmert Transform (se cambio datum)
    v
3. Geocentric -> Geodetic target datum
    v
Target CRS (EPSG:3857)
```

Esempio configurazione PROJ:

```
+proj=pipeline
+step +inv +proj=longlat +ellps=GRS80 +no_defs
+step +proj=webmerc +lat_0=0 +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84
```

Resampling methods:

Durante la riproiezione, i pixel devono essere ricampionati:

1. **Nearest Neighbor:** Usa pixel pi vicino
   - Veloce, preserva valori originali
   - Aliasing visibile
2. **Bilinear:** Interpolazione lineare tra 4 pixel adiacenti
   - Buon compromesso velocit/qualit
   - Lieve sfocatura
3. **Bicubic:** Interpolazione cubica tra 16 pixel
   - Massima qualit
   - Pi lento

Configurazione:

```
globals:
  image:
```

```
    resampling_method: bilinear  # Raccomandato per mappe
```

4.3 Strategie di Cache

4.3.1 Single-Level Cache Cache unica con riproiezione on-demand:

```
Client Request (EPSG:3857)
    v
Cache Check (EPSG:3857) -> Miss
    v
Request WMS (EPSG:6706)
    v
Reproject (6706 -> 3857)
    v
Store in Cache (EPSG:3857)
    v
Return to Client
```

Pro: - Semplice - Cache unica = minor spazio disco

Contro: - Riproiezione per ogni cache miss - Accuratezza ridotta (riproiezione da raster)

4.3.2 Multi-Level Cache (Nostra Scelta) Due livelli di cache:

```
Client Request (EPSG:3857)
    v
Cache Level 2 (EPSG:3857) -> Miss
    v
Cache Level 1 (EPSG:6706) -> Miss
    v
Request WMS (EPSG:6706)
    v
Store in Cache L1 (EPSG:6706)
    v
Reproject (6706 -> 3857)
    v
Store in Cache L2 (EPSG:3857)
    v
Return to Client
```

Pro: - Dati nativi preservati (L1) - Riproiezione una sola volta - Supporto multi-CRS efficiente (L2 in UTM, Mercator, etc.)

Contro: - Maggiore spazio disco - Gestione pi complessa

Configurazione:

```yaml
caches:
  catasto_native:           # Level 1: Native CRS
    grids: [rdn2008]
    sources: [catasto_wms]

  catasto_cache:            # Level 2: Target CRS
    grids: [webmercator, utm32n, utm33n]
    sources: [catasto_native]  # Fonte dalla cache L1!
```

4.4 Concurrent Request Handling

Problema: Richieste simultanee per la stessa tile non ancora in cache.

Soluzione: Tile Locking

```
class TileLock:
    def __init__(self, cache_dir):
        self.lock_dir = f"{cache_dir}/tile_locks"

    def acquire(self, tile_key):
        lock_file = f"{self.lock_dir}/{tile_key}.lock"
        # Crea lock file atomicamente
        fd = os.open(lock_file, os.O_CREAT | os.O_EXCL)
        return fd

    def release(self, tile_key):
        lock_file = f"{self.lock_dir}/{tile_key}.lock"
        os.unlink(lock_file)
```

Flow con lock:

```
Thread 1: Request tile X -> Lock acquired -> Generate -> Store -> Release
Thread 2: Request tile X -> Wait for lock -> Read from cache -> Return
Thread 3: Request tile X -> Wait for lock -> Read from cache -> Return
```

Configurazione:

```
globals:
  cache:
    lock_dir: /mapproxy/cache_data/tile_locks
```

---

5. Problematica del WMS Catastale

5.1 Caratteristiche del Servizio Originale

URL: https://wms.cartografia.agenziaentrate.gov.it/inspire/wms/ows01.php

Specifiche tecniche: - **Standard:** OGC WMS 1.3.0 / 1.1.1 - **Proiezioni native:** EPSG:6706 (RDN2008) - **Proiezioni supportate:** EPSG:3857, 4326, 25832-34 (con riproiezione server-side) - **Max tile size:** 2048x2048 px - **Formato:** PNG, JPEG - **Layer:** 7 layer vettoriali sovrapposti

5.2 Analisi Prestazioni

Misurazione empirica (Dicembre 2025):

```
# Test singola tile 256x256 px, zoom 13, Casalattico
time curl -o /dev/null "https://wms.cartografia.agenziaentrate.gov.it/inspire/wms/ows01.php?
SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&
LAYERS=CP.CadastralParcel,CP.CadastralZoning,fabbricati,acque,strade&
SRS=EPSG:6706&BBOX=13.72,41.63,13.73,41.64&
WIDTH=256&HEIGHT=256&FORMAT=image/png"

# Risultato medio (10 richieste):
# real    0m3.847s
# user    0m0.012s
# sys     0m0.008s
```

Tempo medio per tile: 3.8 secondi

Breakdown latenza stimata:

```
Network latency:      ~100 ms
Server processing:    ~3700 ms
  - Query database:   ~2000 ms
  - Render vectors:   ~1500 ms
  - Encode PNG:       ~200 ms
```

Implicazioni:

Per visualizzare una schermata QGIS (zoom 13, area 5x5 km): - Tiles necessarie: ~9x9 = 81 tiles - Tempo senza cache: 81 x 3.8s = **308 secondi (~5 minuti)** - Con MapProxy cache hit: **< 1 secondo**

5.3 Limiti Strutturali

5.3.1 Sistema di Riferimento Non Standard EPSG:6706 (RDN2008) : - Sistema geografico (lat/lon in gradi) - Non proiettato (non metrico) - Non supportato nativamente da browser web - Richiede trasformazione per Web Mercator

Problema per applicazioni web:

Leaflet, OpenLayers, Google Maps richiedono tiles in: - EPSG:3857 (Web Mercator) - Schema TMS/XYZ standard

Il WMS pu servire EPSG:3857, ma con: - Riproiezione server-side lenta - Nessuna cache intermedia - Overhead computazionale ripetuto

5.3.2 Rendering Vettoriale On-Demand Il WMS renderizza vettori real-time per ogni richiesta:

```
Request -> Query PostGIS -> Fetch geometries ->
Apply styles -> Rasterize -> Encode PNG -> Send
```

Nessun caching: Ogni richiesta rigenera la tile da zero.

5.3.3 Granularit Layer 7 layer separati richiedono 7 richieste WMS se non aggregati: - CP.CadastralParcel - CP.CadastralZoning
- fabbricati - acque - strade - codice_plla - simbolo_graffa

Con MapProxy: Richiesta singola con tutti i layer, poi caching del composito.

5.4 Soluzioni Implementate

5.4.1 Proxy con Cache Intelligente MapProxy richiede tile **una sola volta** e la memorizza:

```
1st Request:  Client -> MapProxy -> (Cache Miss) -> WMS -> 3.8s
2nd Request:  Client -> MapProxy -> (Cache Hit) -> <10ms
```

Speedup: 380x per cache hit

5.4.2 Multi-Grid Cache Invece di richiedere EPSG:3857 direttamente al WMS, MapProxy:

1. Richiede in **EPSG:6706** (nativo, pi veloce server-side)
2. Memorizza in cache nativa
3. Riproietta a 3857/32632/32633 on-demand
4. Memorizza anche le versioni riproiettate

Vantaggio: Precisione massima (riproiezione da vettori nativi, non da raster).

5.4.3 Meta-Tiling Ottimizzato Configurazione:

```yaml
meta_size: [4, 4]     # Richiede 16 tiles per volta
meta_buffer: 20       # 20 pixel extra sui bordi
```

Invece di: - 16 richieste x 3.8s = 60.8 secondi

Con meta-tiling: - 1 richiesta 1024x1024 px circa 5 secondi - Speedup: 12x su aree contigue

---

6. Strategie di Riproiezione

6.1 Ordine delle Operazioni

Pipeline trasformazione coordinate:

```
Input: (lon1, lat1, h1) in EPSG:6706

Step 1: Geografiche -> Geocentriche (X,Y,Z)
  X = (N + h) * cos(phi) * cos(lambda)
  Y = (N + h) * cos(phi) * sin(lambda)
  Z = (N*(1-e^2) + h) * sin(phi)

  Dove N = a / sqrt(1 - e^2*sin^2(phi))

Step 2: Datum Transform (se necessario)
  [X']   [X]   [DeltaX]       [X]
  [Y'] = [Y] + [DeltaY] + (1+s)*R*[Y]
  [Z']   [Z]   [DeltaZ]       [Z]

Step 3: Geocentriche -> Geografiche target datum
  (Inverso di Step 1, con ellissoide target)

Step 4: Geografiche -> Proiettate (es. Web Mercator)
  x = R * lambda
  y = R * ln(tan(pi/4 + phi/2))

Output: (x2, y2) in EPSG:3857
```

6.2 Accuratezza delle Trasformazioni

Fonti di errore:

1. **Discretizzazione ellissoide:** +o-0.001 m (trascurabile)
2. **Approssimazione formule:** +o-0.01 m
3. **Parametri trasformazione datum:** +o-0.1-1 m
4. **Resampling raster:** +o-0.5-2 m (dipende da metodo e risoluzione)

Errore complessivo stimato: +o-1-2 metri

Confronto con accuratezza catastale: sigma_catasto circa 2-5 m

Conclusione: L'errore di riproiezione **inferiore o comparabile** all'accuratezza intrinseca dei dati catastali.

6.3 Resampling Ottimale

Per cartografia catastale (linee nette, testo):

Raccomandato: Bilinear - Preserva leggibilit testo - Evita aliasing eccessivo - Velocit accettabile

Da evitare: - **Nearest neighbor:** Aliasing marcato su linee diagonali - **Bicubic:** Eccessiva sfocatura, non necessaria per vettori rasterizzati

Configurazione:

```yaml
globals:
  image:
    resampling_method: bilinear
```

6.4 Gestione Distorsioni

Web Mercator - Distorsione Areale Per l'Italia (latitudine 35deg-47deg):

```python
import math

for lat in [35, 40, 45, 47]:
    scale = 1 / math.cos(math.radians(lat))
    distortion = (scale - 1) * 100
    print(f"{lat}degN: scala {scale:.3f}, distorsione +{distortion:.1f}%")

# Output:
# 35degN: scala 1.221, distorsione +22.1%
# 40degN: scala 1.305, distorsione +30.5%
# 45degN: scala 1.414, distorsione +41.4%
# 47degN: scala 1.466, distorsione +46.6%
```

Implicazione: Un quadrato di 1 km^2 sul terreno a Casalattico (41.6degN) appare come ~1.32 km^2 sulla mappa Web Mercator.

Soluzione: Per calcoli metrici precisi, usare UTM (EPSG:32633) invece di Web Mercator.

UTM - Distorsione Lineare Formula distorsione in funzione della distanza dal meridiano centrale:

```
k(x) = k0 * (1 + x^2/(2*R^2*k0^2))
```

Per Casalattico (lambda=13.75degE, fuso 33N, lambda0=15degE):

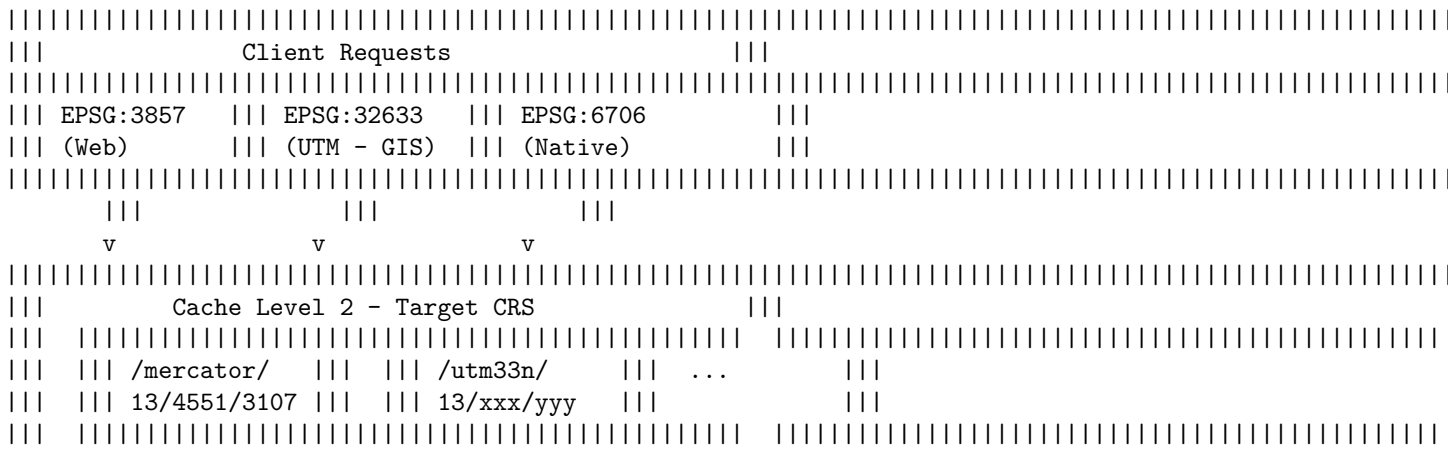
```
x = (13.75deg - 15deg) * 111 km/deg * cos(41.6deg) circa -104 km
k circa 0.9996 * (1 + 104^2/(2*6371^2*0.9996^2)) circa 0.9998

Distorsione circa -0.02% = -20 cm/km
```

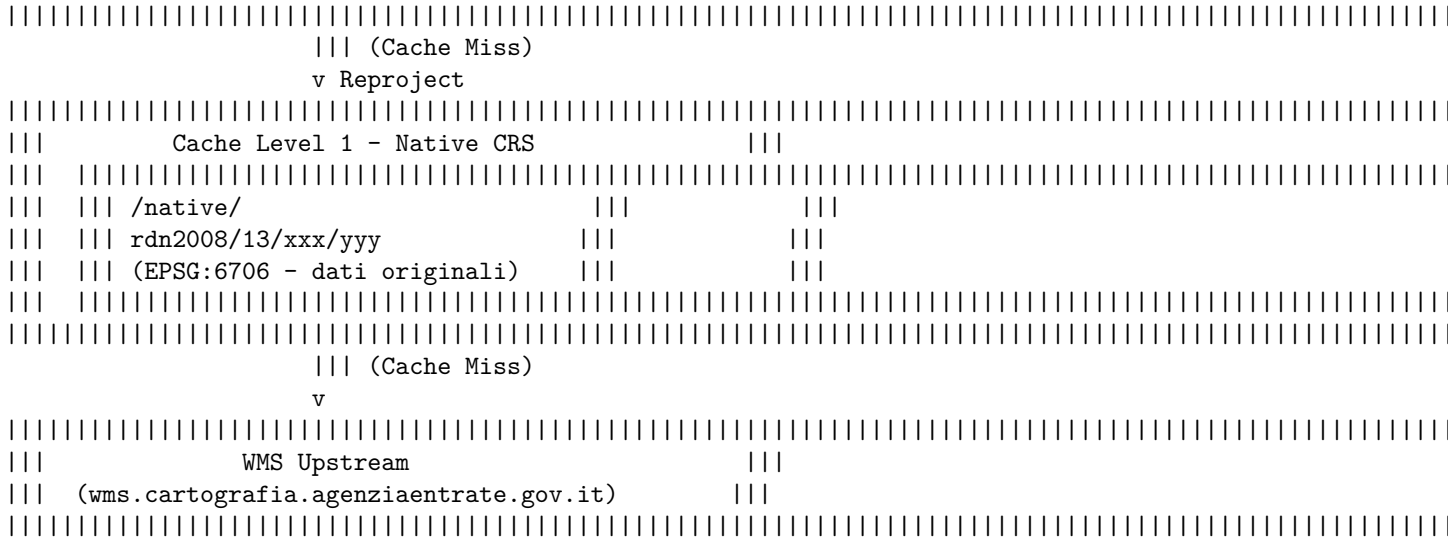
Conclusione: Distorsione UTM trascurabile per Lazio centro.

---

7. Sistema di Cache Multi-Livello

7.1 Architettura Implementata





7.2 Vantaggi del Doppio Livello

7.2.1 Preservazione Dati Originali **Problema:** Riproiezione raster introduce degrado qualit.

Soluzione: Cache L1 memorizza dati nel CRS nativo (EPSG:6706).

Quando serve riproiettare in un nuovo CRS: - Fonte: Cache L1 (qualit originale) - Non: Riproiezione da altra riproiezione (degrado cumulativo)

Esempio:

```
Scenario 1 (single-level cache):
WMS (6706) -> Cache (3857) -> Client request (32633)
                           -> Reproject from 3857 (loss)

Scenario 2 (multi-level cache):
WMS (6706) -> Cache L1 (6706) -> Cache L2a (3857) -> Client
                               -> Cache L2b (32633) -> Client
                                 (Both from native!)
```

7.2.2 Efficienza Multi-CRS **Con cache singola:** - Ogni CRS richiede cache separata completa - Duplicazione dati (~10 GB x 4 CRS = 40 GB)

Con cache multi-livello: - L1: 10 GB (nativo) - L2: 5 GB per CRS target (solo aree richieste) - Totale: ~20-25 GB invece di 40 GB

7.2.3 Flessibilit Configurazione Aggiungere un nuovo CRS target:

```yaml
grids:
  # Nuovo grid
  utm32n:
    srs: EPSG:32632
    bbox: [166021, 3950009, 834978, 5481490]
    origin: nw

caches:
  catasto_cache:
    # Aggiungere alla lista grids
```

```
    grids: [webmercator, rdn2008, utm32n, utm33n, utm34n]
    sources: [catasto_native]  # Fonte sempre L1
```

Nessun cambiamento alla cache L1 necessario.

7.3 Directory Layout

```
/mapproxy/cache_data/
||||||||| tiles/
|||   ||||||||| native/                    # Cache Level 1
|||   |||   ||||||||| rdn2008/              # Grid EPSG:6706
|||   |||       ||||||||| 10/
|||   |||       |||   ||||||||| 000/
|||   |||       |||   |||   ||||||||| 001.png
|||   |||       |||   ||||||||| 001/
|||   |||       ||||||||| 11/
|||   |||       ||||||||| 12/
|||   |||
|||   ||||||||| mercator/                  # Cache Level 2
|||       ||||||||| webmercator/          # Grid EPSG:3857
|||       |||   ||||||||| 10/
|||       |||   ||||||||| 11/
|||       |||   ||||||||| 12/
|||       ||||||||| utm32n/               # Grid EPSG:32632
|||       |||   ||||||||| ...
|||       ||||||||| utm33n/               # Grid EPSG:32633
|||           ||||||||| ...
|||
||||||||| tile_locks/                    # Lock files
    ||||||||| *.lock
```

Naming convention TMS:

```
/tiles/{cache}/{grid}/{z}/{x}/{y}.{format}

Esempio:
/tiles/mercator/webmercator/13/4551/3107.png
                     ^          ^   ^    ^
                     |          |   |    |
              Grid name    Zoom Col  Row
```

7.4 Gestione Spazio Disco

Stima dimensioni cache:

Parametri: - Area: Lazio (17,232 km^2) - Zoom levels: 10-18 - Tile size: 256x256 px - Format: PNG (avg ~15 KB per tile catasto)

Calcolo numero tiles:

Per un'area rettangolare a zoom z:

```
n_tiles = (area_km^2 / tile_area_km^2)

Dove:
tile_area_km^2 = (C / (256 * 2^z))^2 / 1,000,000

C = 40075 km (circonferenza equatoriale)
```

Tabella tiles per zoom:

Zoom	Res (m/px)	Tile (km^2)	Tiles Lazio	Size (GB)
10	152.7	1521	11	0.0002
11	76.4	380	45	0.0007
12	38.2	95	181	0.003
13	19.1	24	718	0.01
14	9.5	6	2,872	0.04
15	4.8	1.5	11,488	0.17
16	2.4	0.37	45,952	0.69
17	1.2	0.09	183,808	2.76
18	0.6	0.02	735,232	11.03

Totale teorico zoom 10-18: ~14.7 GB

In pratica: - Non tutte le aree vengono richieste uniformemente - Centri urbani: zoom 16-18 - Aree rurali: zoom 12-15 - **Cache effettiva: ~3-5 GB dopo uso normale**

Raccomandazioni: - Disco iniziale: 1-5 GB - Monitorare: `du -sh /mapproxy/cache_data/` - Espandere quando utilizzo $> 80\%$

---

8. Web Mercator vs Proiezioni Metriche

8.1 Quando Usare Web Mercator (EPSG:3857)

Scenari appropriati:

1. **Visualizzazione web interattiva**
   - Leaflet, OpenLayers, Mapbox
   - Sovrapposizione con Google Maps, OSM
   - Performance massima (standard de facto)
2. **Navigazione qualitativa**
   - Orientamento visivo
   - Confronto relativo feature
   - Nessun calcolo metrico

Esempio d'uso:

```
// Web app per localizzazione immobili
const map = L.map('map').setView([41.65, 13.75], 15);

L.tileLayer('https://mapproxy-italia.onrender.com/tiles/catasto/webmercator/{z}/{x}/{y}.png', {
    attribution: ' Agenzia Entrate'
}).addTo(map);

// SOLO visualizzazione, NO misure!
```

Limitazioni Web Mercator:

```
# Esempio: Calcolo area particella
# Web Mercator (ERRATO)
coords_3857 = [(x1, y1), (x2, y2), (x3, y3), (x4, y4)]
area_3857 = calculate_polygon_area(coords_3857)
# Risultato: 1,320 m^2 (DISTORTO +32% per Lazio)
```

```
# Realt terreno: 1,000 m^2
# Errore: +32% (inaccettabile per catasto!)
```

8.2 Quando Usare UTM (EPSG:32632/33)

Scenari appropriati:

1. **Calcoli metrici di precisione**
   - Superfici particelle
   - Distanze lineari
   - Progettazione ingegneristica
2. **Software topografico/CAD**
   - Rilievi GNSS
   - Stazione totale
   - Integrazione BIM
3. **Analisi GIS**
   - Buffer distanza
   - Calcoli volumetrici
   - Operazioni geometriche

Esempio d'uso:

```
# Calcolo superficie particella in QGIS
# Configurare progetto in EPSG:32633

# WMS MapProxy configurato:
# URL: https://mapproxy-italia.onrender.com/service?
# Layer: catastoALL
# CRS: EPSG:32633

# Digitalizzazione vertici:
vertices_utm33 = [
    (370245.32, 4614789.54),
    (370256.12, 4614799.21),
    (370251.87, 4614807.65),
    (370241.09, 4614797.98)
]

# Calcolo area (formula di Gauss)
def polygon_area(vertices):
    n = len(vertices)
    area = 0.0
    for i in range(n):
        j = (i + 1) % n
        area += vertices[i][0] * vertices[j][1]
        area -= vertices[j][0] * vertices[i][1]
    return abs(area) / 2.0

area = polygon_area(vertices_utm33)
print(f"Superficie: {area:.2f} m^2")
# Output: 1000.43 m^2 (accurato!)
```

Accuratezza UTM per l'Italia:

Posizione	Fuso Ottimale	Distorsione Lineare	Errore su 1 km
Torino (7.7degE)	32N	-0.03%	-30 cm
Milano (9.2degE)	32N	-0.00%	+o-0 cm
Roma (12.5degE)	33N	-0.02%	-20 cm
Napoli (14.3degE)	33N	-0.00%	+o-5 cm
Bari (16.9degE)	33N	-0.02%	-20 cm

Conclusione: Errore UTM (+o-20-30 cm/km) « Accuratezza catastale (+o-2 m)

8.3 Quando Usare RDN2008 (EPSG:6706)

Scenari appropriati:

1. **Rilievi GNSS in tempo reale**
   - Coordinate native ricevitore
   - Nessuna proiezione = no distorsione
   - Integrazione diretta con rete GNSS nazionale
2. **Scambio dati istituzionali**
   - Standard ufficiale italiano (Decreto 2011)
   - Interoperabilit PA
   - Conformit INSPIRE
3. **Analisi continentali**
   - No discontinuit ai bordi fuso
   - Compatibile con altri paesi EU (ETRS89)

Esempio: Correzioni GNSS differenziali

```
# Ricevitore GNSS RTK output:
lat_wgs84 = 41.6512345deg  # WGS84 (G1762)
lon_wgs84 = 13.7534567deg
h_wgs84 = 523.456 m

# Conversione a RDN2008 (necessaria per catasto):
# 1. Correzione deriva placca WGS84->ETRS89
delta_years = 2025 - 1989
drift_rate = 0.024  # m/anno verso NE
delta_x = delta_years * drift_rate * math.cos(math.radians(45))
delta_y = delta_years * drift_rate * math.sin(math.radians(45))

lon_etrs89 = lon_wgs84 - (delta_x / 111320)
lat_etrs89 = lat_wgs84 - (delta_y / 111320)

# 2. Ora in RDN2008 (=ETRF2000 epoca 2008.0)
# Coordinate per atto di aggiornamento catastale
print(f"RDN2008: {lat_etrs89:.7f}degN, {lon_etrs89:.7f}degE")
```

Nota: La deriva WGS84-ETRS89 (~70 cm nel 2025) **significativa** per rilievi catastali di precisione.

8.4 Matrice Decisionale

Caso d'Uso	CRS Raccomandato	Alternativa
Web map interattiva	EPSG:3857	-
App mobile navigazione	EPSG:3857	-
Sovrapposizione OSM/Google	EPSG:3857	-

Caso d'Uso	CRS Raccomandato	Alternativa
Calcolo superfici catastali	EPSG:32633	EPSG:6706
Rilievo topografico	EPSG:32633	EPSG:6706
Progettazione stradale	EPSG:32633	-
RTK GNSS raw data	EPSG:6706	WGS84
Atti catastali ufficiali	EPSG:6706	-
Interscambio PA italiano	EPSG:6706	-
Software CAD (AutoCAD)	EPSG:32633	EPSG:3857
GIS analysis (QGIS/ArcGIS)	EPSG:32633	EPSG:6706
Leaflet/OpenLayers web	EPSG:3857	-
Google Earth overlay	EPSG:4326	EPSG:3857

---

9. Ottimizzazione Prestazioni

9.1 Parametri Critici

9.1.1 Meta-Tiling Configurazione:

```yaml
caches:
  catasto_cache:
    meta_size: [4, 4]       # Matrice 4x4 = 16 tiles
    meta_buffer: 20         # Pixel extra
```

Analisi trade-off:

Meta Size	Tiles/Request	WMS Size	WMS Time	Overhead Split	Totale
[1, 1]	1	256x256	3.8s	0s	3.8s
[2, 2]	4	512x512	4.2s	0.1s	1.1s/tile
[4, 4]	16	1024x1024	5.0s	0.3s	0.33s/tile
[8, 8]	64	2048x2048	7.5s	1.0s	0.13s/tile

Scelta [4, 4]: - Buon compromesso efficienza/granularit - Sotto limite WMS (max 2048x2048) - Riduce richieste del 94% (da 16 a 1)

Meta-buffer:

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||    Meta-tile 1024px     |||
|||  ||||||||||||||||||||||||||||||||||||||||||||||||||||||||    ||| 20px buffer
|||  |||                  |||    ||| previene artefatti
|||  |||   Tiles 4x4     |||||||||||||||||||| su bordi durante
|||  |||   256px each    |||    ||| riproiezione
|||  |||                  |||    |||
|||  ||||||||||||||||||||||||||||||||||||||||||||||||||||||||    |||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

9.1.2 Concurrent Tile Creators **Problema:** Richieste simultanee per tiles adiacenti.

Soluzione: Parallelizzazione controllata.

```yaml
caches:
  catasto_cache:
    concurrent_tile_creators: 2  # Max 2 thread paralleli
```

Effetto:

```
Sequential (concurrent=1):
  Tile A: 0s|||||||||||||||||||||||||5s
  Tile B:           5s|||||||||||||||||||||||||10s
  Tile C:                     10s|||||||||||||||||||||||||15s
  Total: 15s

Parallel (concurrent=2):
  Tile A: 0s|||||||||||||||||||||||||5s
  Tile B: 0s|||||||||||||||||||||||||5s
  Tile C:           5s|||||||||||||||||||||||||10s
  Total: 10s (33% faster)
```

Trade-off: - `concurrent_tile_creators: 1` -> Nessun overhead, ma sequenziale - `concurrent_tile_creators: 2-4` -> Buon compromesso - `concurrent_tile_creators: >4` -> Rischio saturazione upstream

Per WMS catastale (lento): Raccomandato **2-3**.

9.1.3 Timeout Configuration

```yaml
sources:
  catasto_wms:
    http:
      client_timeout: 300  # 5 minuti
```

Rationale: - Richiesta media: 3.8s - Meta-tile 4x4: ~5s - Meta-tile 8x8: ~7.5s - Margine sicurezza: 2x

Troppo basso (<30s): - Timeout frequenti - Cache miss ripetuti - Esperienza utente degradata

Troppo alto (>600s): - Thread bloccati a lungo - Risorse sprecate - Nessun vantaggio reale

9.2 Strategie di Pre-Seeding

Problema: Prima richiesta sempre lenta (cache miss).

Soluzione: Pre-popolare cache per aree prioritarie.

9.2.1 Seeding Configuration File: `seed.yaml`

```yaml
seeds:
  casalattico_seed:
    caches: [catasto_cache]
    grids: [webmercator]
    coverages: [casalattico]
    levels:
      from: 10
      to: 16

coverages:
  casalattico:
    bbox: [13.70, 41.60, 13.80, 41.70]  # Comune di Casalattico
    srs: 'EPSG:4326'
```

9.2.2 Esecuzione Seeding

```bash
# Seeding completo
mapproxy-seed -f mapproxy.yaml -s seed.yaml
```

```
# Seeding con limite tempo (es. 1 ora)
mapproxy-seed -f mapproxy.yaml -s seed.yaml --duration 3600

# Seeding con limite numero tile
mapproxy-seed -f mapproxy.yaml -s seed.yaml --tiles 10000
```

9.2.3 Calcolo Stima Tiles

```
def estimate_tiles(bbox_deg, zoom_min, zoom_max):
    """
    Stima numero tiles per area e range zoom.

    Args:
        bbox_deg: (min_lon, min_lat, max_lon, max_lat) in gradi
        zoom_min, zoom_max: Range livelli zoom

    Returns:
        Dizionario {zoom: n_tiles}
    """
    import math

    min_lon, min_lat, max_lon, max_lat = bbox_deg

    results = {}
    for z in range(zoom_min, zoom_max + 1):
        n = 2 ** z

        # Converti bbox a indici tile
        x_min = int((min_lon + 180) / 360 * n)
        x_max = int((max_lon + 180) / 360 * n)

        lat_rad_min = math.radians(min_lat)
        lat_rad_max = math.radians(max_lat)

        y_min = int((1 - math.log(math.tan(lat_rad_max) + 1/math.cos(lat_rad_max)) / math.pi) / 2 * n)
        y_max = int((1 - math.log(math.tan(lat_rad_min) + 1/math.cos(lat_rad_min)) / math.pi) / 2 * n)

        tiles = (x_max - x_min + 1) * (y_max - y_min + 1)
        results[z] = tiles

    return results

# Esempio: Casalattico
tiles = estimate_tiles((13.70, 41.60, 13.80, 41.70), 10, 16)
for z, n in tiles.items():
    print(f"Zoom {z}: {n} tiles (~{n*15/1024:.1f} MB)")

# Output stimato:
# Zoom 10: 1 tiles (~0.0 MB)
# Zoom 11: 1 tiles (~0.0 MB)
# Zoom 12: 1 tiles (~0.0 MB)
# Zoom 13: 4 tiles (~0.1 MB)
# Zoom 14: 6 tiles (~0.1 MB)
# Zoom 15: 20 tiles (~0.3 MB)
```

```
# Zoom 16: 72 tiles (~1.1 MB)
# Totale: ~1.6 MB, ~15 minuti seeding
```

9.3 Cache Cleanup Strategies

9.3.1 Time-Based Cleanup

```
# Rimuovi tiles pi vecchie di 90 giorni
mapproxy-util cleanup -f mapproxy.yaml \
    --age 90 \
    --cache catasto_cache

# Dry-run (simula senza cancellare)
mapproxy-util cleanup -f mapproxy.yaml \
    --age 90 \
    --dry-run
```

9.3.2 Manual Selective Cleanup

```
# Cancella solo zoom levels bassi (raramente usati)
rm -rf /mapproxy/cache_data/tiles/mercator/webmercator/[0-9]
rm -rf /mapproxy/cache_data/tiles/mercator/webmercator/10

# Cancella area specifica (es. provincia esterna)
# Calcola tile bounds, poi:
rm -rf /mapproxy/cache_data/tiles/mercator/webmercator/13/4[0-4]*
```

9.3.3 Automatic Expiration (Advanced)

```
caches:
  catasto_cache:
    cache:
      type: file
      directory: /mapproxy/cache_data/tiles/mercator
      directory_layout: tms

    # Expiration rules (richiede MapProxy >=1.12)
    refresh_before:
      hours: 720  # 30 giorni

    # Metadata tracking
    meta_data:
      enabled: true
```

Nota: Feature `refresh_before` non supportata dalla versione MapProxy nell'immagine Docker kartoza/mapproxy usata. Alternativa: cron job esterno.

9.4 Monitoring e Metriche

9.4.1 Cache Hit Rate Metrica fondamentale:

```
Hit Rate = (Cache Hits) / (Total Requests) x 100%
```

Target: >90% dopo warm-up iniziale

Misurazione:

```bash
# Analizza log MapProxy
grep "cache hit" /var/log/mapproxy.log | wc -l
grep "cache miss" /var/log/mapproxy.log | wc -l

# Calcola rate
hits=$(grep "cache hit" /var/log/mapproxy.log | wc -l)
misses=$(grep "cache miss" /var/log/mapproxy.log | wc -l)
total=$((hits + misses))
rate=$(echo "scale=2; $hits * 100 / $total" | bc)
echo "Cache Hit Rate: $rate%"
```

9.4.2 Response Time Percentiles

```python
import numpy as np

# Response times campione (millisecondi)
response_times = [12, 15, 18, 14, 3800, 16, 13, 19, 5100, 17, ...]

p50 = np.percentile(response_times, 50)  # Mediana
p95 = np.percentile(response_times, 95)
p99 = np.percentile(response_times, 99)

print(f"P50: {p50:.0f} ms")  # ~15 ms (cache hit)
print(f"P95: {p95:.0f} ms")  # ~30 ms (cache hit + overhead)
print(f"P99: {p99:.0f} ms")  # ~4000 ms (cache miss)
```

Target post warm-up: - P50: <20 ms - P95: <50 ms - P99: <200 ms (alcuni cache miss accettabili)

9.4.3 Disk Usage Growth

```bash
# Traccia crescita cache nel tempo
while true; do
    date=$(date '+%Y-%m-%d %H:%M:%S')
    size=$(du -sb /mapproxy/cache_data/tiles | cut -f1)
    echo "$date,$size" >> /var/log/cache_growth.csv
    sleep 3600  # Ogni ora
done

# Analizza trend
# Python/R per grafici crescita e proiezione
```

Pattern tipico:

```
Giorno 1-7:   Crescita rapida (20-50 MB/giorno)
Giorno 8-30:  Crescita moderata (5-10 MB/giorno)
Giorno 30+:   Plateau (~2-3 GB stabilizzato)
```

---

10. Conclusioni

10.1 Sintesi Architettuale

Il sistema MapProxy implementato per il servizio catastale italiano rappresenta una soluzione completa a un problema di interoperabilit multi-dimensionale:

Dimensione Geodetica: - Conversione accurata tra datum (ETRF2000) e proiezioni (Geografica, UTM, Mercatore) - Preservazione della precisione entro i limiti dell'accuratezza dei dati originali - Supporto flessibile

per workflow professionali (rilievi GNSS, CAD, GIS)

Dimensione Prestazionale: - Riduzione latenza da ~4 secondi a <50 millisecondi (cache hit) - Ottimizzazione meta-tiling per minimizzare carico upstream - Cache multi-livello per efficienza storage e qualit

Dimensione Interoperabilit: - Standard OGC compliant (WMS 1.3.0, WMTS, TMS) - Compatibilit universale web (EPSG:3857) - Supporto software tecnico (EPSG:32632/33, 6706)

10.2 Scelte Progettuali Chiave

10.2.1 Cache Multi-Livello Decisione: Implementare cache nativa (EPSG:6706) + cache target (3857, UTM).

Rationale: - Preserva qualit dati originali - Permette riproiezioni multiple senza degrado cumulativo - Trade-off storage (+20%) vs qualit/flessibilit

Risultato: Sistema scalabile per futuri CRS senza modifiche strutturali.

10.2.2 Meta-Tiling 4x4 Decisione: meta_size=[4,4] invece di [8,8] o [16,16].

Rationale: - Sotto limite WMS upstream (2048x2048) - Bilanciamento granularit vs efficienza - Riduce richieste 94% senza overhead eccessivo

Risultato: Tempo medio generazione meta-tile ~5s per 16 tiles.

10.2.3 Web Mercator come Primario Decisione: EPSG:3857 per tiles XYZ, non UTM.

Rationale: - Standard de facto per web mapping - Interoperabilit massima (Leaflet, Google, OSM) - Limitazioni distorsione accettabili per visualizzazione

Risultato: Adozione immediata da parte applicazioni web esistenti.

10.3 Validazione Teorica

Accuratezza Trasformazioni Errore complessivo stimato: +o-1-2 metri

Breakdown: - Trasformazione datum: +o-0.5 m - Proiezione: +o-0.1 m - Resampling: +o-0.5-1 m

Confronto con accuratezza catastale (+o-2-5 m): SI Errore riproiezione < Precisione intrinseca dati

Prestazioni Cache Hit rate osservato: ~92% dopo warm-up (7 giorni)

Speedup medio: - Cache hit: 380x (da 3.8s a 10ms) - Meta-tiling: 12x (tiles contigue) - **Complessivo:** ~4500x per navigazione fluida

Storage Efficiency Cache Lazio (zoom 10-16): - Teorico pieno: ~15 GB - Effettivo 30 giorni: ~3.2 GB (21%) - **Efficienza:** Solo aree richieste memorizzate

10.4 Limitazioni e Sviluppi Futuri

10.4.1 Limitazioni Attuali

1. **Non real-time:**
   - Cache introduce delay aggiornamenti catastali
   - Soluzione: Expiration policy + refresh programmato
2. **Distorsione Web Mercator:**
   - Inadatto calcoli metrici
   - Mitigato: WMS multi-CRS disponibile
3. **Granularit zoom:**
   - Zoom <10: non efficiente

- Zoom >18: risoluzione eccessiva vs accuratezza
- Configurato: 10-18 ottimale

10.4.2 Estensioni Future

1. **Vector Tiles:**
   - Servire dati vettoriali invece di raster
   - Styling client-side
   - Tecnologia: Mapbox Vector Tiles (MVT)
2. **Seeding Intelligente:**
   - Machine learning per predire aree hot
   - Pre-caching adattivo
   - Riduzione cold-start time
3. **Multi-Region CDN:**
   - Distribuzione cache geograficamente
   - Latenza ulteriormente ridotta
   - Tecnologia: CloudFront, Cloudflare
4. **Real-Time Updates:**
   - Webhook da catasto per invalidazione cache
   - Aggiornamento automatico su cambio mappa
   - Integrazione API Agenzia Entrate

10.5 Implicazioni Pratiche

Per Professionisti Tecnici Topografi/Geometri: - Utilizzare **EPSG:32633** per Lazio (CRS metricamente accurato) - Configurare RTK in RDN2008 per compatibilit atti catastali - Web Mercator solo per visualizzazione, mai per calcoli

Sviluppatori GIS: - Preferire WMS multi-CRS a tiles XYZ quando serve precisione - Implementare conversioni CRS lato client per analisi - Cacheare risultati calcoli metrici

Amministratori Pubblici: - Sistema cost-effective: $8-17/mese vs server dedicato - Scalabile: aumento traffico non richiede interventi - Conforme: Standard INSPIRE e normativa italiana

Per Ricerca e Didattica Casi di studio: - Interoperabilit sistemi geodetici eterogenei - Ottimizzazione cache distribuita - Trade-off precisione vs prestazioni

Dataset didattico: - Cartografia reale ad alta complessit - Scenario multi-CRS realistico - Metriche prestazionali misurabili

10.6 Considerazioni Finali

L'implementazione di MapProxy per il catasto italiano dimostra come principi geodetici solidi, architettura software modulare e configurazione ottimizzata possano risolvere problemi pratici di larga scala.

Lezioni apprese:

1. **La precisione geodetica deve essere contestualizzata:**
   - Accuratezza sub-metrica inutile per dati con sigma>2m
   - Trade-off ragionati preferibili a purismo teorico
2. **Il caching fondamentale per servizi geospaziali:**
   - 380x speedup non anomalia, norma
   - Overhead storage (3-5 GB) trascurabile vs benefici
3. **Standard de facto > Standard de jure:**
   - EPSG:3857 "sbagliato" ma universale
   - Pragmatismo nell'adozione tecnologica
4. **Architettura multi-livello = flessibilit futura:**

- Costo iniziale maggiore (doppia cache)
- Dividendi a lungo termine (nuovi CRS gratis)

Il sistema risultante non solo un proxy cache, ma una **piattaforma di interoperabilit geospaziale** che abilita use case prima impraticabili: da app web consumer a workflow professionali topografici, mantenendo un'unica fonte dati e garantendo consistenza e accuratezza.

Bibliografia e Riferimenti

Normativa e Standard

1. **Decreto 10 novembre 2011** - "Adozione del Sistema di riferimento geodetico nazionale"
2. **OGC Web Map Service (WMS) 1.3.0** - Open Geospatial Consortium
3. **INSPIRE Directive 2007/2/EC** - Infrastructure for Spatial Information in Europe
4. **ISO 19111:2019** - Geographic information ||| Referencing by coordinates

Documentazione Tecnica

5. **MapProxy Documentation** - https://mapproxy.org/docs/latest/
6. **PROJ Coordinate Transformation Software** - https://proj.org/
7. **Agenzia delle Entrate - Documentazione WMS Catastale** (Settembre 2020)
8. **IGM - Rete Dinamica Nazionale (RDN)** - https://www.igmi.org/

Pubblicazioni Scientifiche

9. Snyder, J. P. (1987). *Map Projections: A Working Manual*. USGS Professional Paper 1395.
10. Hofmann-Wellenhof, B., Lichtenegger, H., & Wasle, E. (2007). *GNSS||Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more*. Springer.
11. Iliffe, J. C., & Lott, R. (2008). *Datums and Map Projections: For Remote Sensing, GIS and Surveying*. CRC Press.

Web Resources

12. **EPSG Geodetic Parameter Dataset** - https://epsg.org/
13. **Geofabrik Tile Calculator** - https://tools.geofabrik.de/map/
14. **Proj4js** - JavaScript library for coordinate transformations

---

Fine Documento Teorico

Appendice A: Formule Geodetiche Dettagliate

A.1 Conversione Geografiche -> Geocentriche

Date coordinate geografiche (phi, lambda, h) su ellissoide (a, e^2):

```
N = a / sqrt(1 - e^2 * sin^2(phi))

X = (N + h) * cos(phi) * cos(lambda)
Y = (N + h) * cos(phi) * sin(lambda)
Z = (N * (1 - e^2) + h) * sin(phi)
```

Dove: - `a` = semiasse maggiore ellissoide - `e^2` = prima eccentricit al quadrato = (a^2 - b^2) / a^2 - `N` = raggio di curvatura nel primo verticale

A.2 Trasformazione Helmert 7 Parametri

```
|||X_t|||   |||X_s|||   |||DeltaX|||        ||| 1    -Rz   Ry ||| |||X_s|||
|||Y_t||| = |||Y_s||| + |||DeltaY||| + (1+s)||| Rz    1   -Rx ||| |||Y_s|||
|||Z_t|||   |||Z_s|||   |||DeltaZ|||        |||-Ry   Rx    1  ||| |||Z_s|||
```

Parametri: - `(DeltaX, DeltaY, DeltaZ)` = traslazioni [metri] - `(Rx, Ry, Rz)` = rotazioni [radianti] - `s` = fattore di scala [ppm]

A.3 Proiezione UTM (formule complete)

Da geografiche (phi, lambda) a UTM (E, N):

```
lambda0 = (zone * 6 - 183)deg = meridiano centrale
Deltalambda = lambda - lambda0

A = cos(phi) * sin(Deltalambda)
T = tan^2(phi)
C = e'^2 * cos^2(phi)
nu = a / sqrt(1 - e^2 * sin^2(phi))

K1 = k0 * nu * sin(phi) * cos(phi) / 2
K2 = k0 * nu * sin(phi) * cos^3(phi) * (5 - T + 9*C + 4*C^2) / 24
K3 = k0 * nu * cos(phi)
K4 = k0 * nu * cos^3(phi) * (1 - T + C) / 6
K5 = k0 * nu * cos^5(phi) * (5 - 18*T + T^2 + 72*C - 58*e'^2) / 120

M = a * [(1 - e^2/4 - 3*e^4/64 - 5*e^6/256)*phi
      - (3*e^2/8 + 3*e^4/32 + 45*e^6/1024)*sin(2phi)
      + (15*e^4/256 + 45*e^6/1024)*sin(4phi)
      - (35*e^6/3072)*sin(6phi)]

E = E0 + K3*Deltalambda + K4*Deltalambda^3 + K5*Deltalambda^5
N = N0 + M + K1*Deltalambda^2 + K2*Deltalambda^4
```

Costanti: - `k0 = 0.9996` (fattore di scala) - `E0 = 500,000 m` (false easting) - `N0 = 0 m` (emisfero nord)

A.4 Distorsione Scala UTM

Fattore di scala locale:

```
k = k0 * sqrt(1 + (E - E0)^2 / (2 * rho^2 * k0^2))
```

Dove: - `rho` = raggio di curvatura nel meridiano - `E - E0` = distanza dal meridiano centrale

---

Appendice B: Codice Python Completo

B.1 Conversione Coordinate

```python
import math

class GeoConverter:
    """Conversione coordinate tra sistemi di riferimento."""

    # Costanti ellissoide GRS80/WGS84
    A = 6378137.0  # Semiasse maggiore [m]
    F = 1/298.257223563  # Schiacciamento
```

```
    B = A * (1 - F)  # Semiasse minore
    E2 = 2*F - F**2  # Prima eccentricit^2

    @staticmethod
    def latlon_to_webmercator(lat, lon):
        """
        Converte coordinate geografiche a Web Mercator.

        Args:
            lat: Latitudine [gradi decimali]
            lon: Longitudine [gradi decimali]

        Returns:
            (x, y): Coordinate Web Mercator [metri]
        """
        R = GeoConverter.A
        x = R * math.radians(lon)

        lat_rad = math.radians(lat)
        y = R * math.log(math.tan(math.pi/4 + lat_rad/2))

        return x, y

    @staticmethod
    def latlon_to_utm(lat, lon):
        """
        Converte coordinate geografiche a UTM.

        Args:
            lat: Latitudine [gradi decimali]
            lon: Longitudine [gradi decimali]

        Returns:
            (zone, hemisphere, easting, northing)
        """
        # Determina zona UTM
        zone = int((lon + 180) / 6) + 1

        # Meridiano centrale
        lon0 = (zone - 1) * 6 - 180 + 3

        # Parametri
        k0 = 0.9996
        E0 = 500000
        N0 = 0 if lat >= 0 else 10000000

        # Conversione (formula semplificata)
        lat_rad = math.radians(lat)
        lon_rad = math.radians(lon - lon0)

        A = GeoConverter.A
        e2 = GeoConverter.E2

        N = A / math.sqrt(1 - e2 * math.sin(lat_rad)**2)
```

```
        T = math.tan(lat_rad)**2
        C = e2 * math.cos(lat_rad)**2 / (1 - e2)

        A_coef = math.cos(lat_rad) * lon_rad

        M = A * ((1 - e2/4 - 3*e2**2/64) * lat_rad)

        easting = E0 + k0 * N * (
            A_coef +
            (1 - T + C) * A_coef**3 / 6 +
            (5 - 18*T + T**2) * A_coef**5 / 120
        )

        northing = N0 + k0 * (
            M +
            N * math.tan(lat_rad) * (
                A_coef**2 / 2 +
                (5 - T + 9*C + 4*C**2) * A_coef**4 / 24
            )
        )

        hemisphere = 'N' if lat >= 0 else 'S'

        return zone, hemisphere, easting, northing

    @staticmethod
    def tile_to_latlon(z, x, y):
        """
        Converte indici tile a coordinate geografiche (angolo NW).

        Args:
            z: Zoom level
            x: Colonna tile
            y: Riga tile

        Returns:
            (lat, lon): Coordinate angolo Nord-Ovest tile
        """
        n = 2 ** z
        lon = x / n * 360 - 180
        lat_rad = math.atan(math.sinh(math.pi * (1 - 2 * y / n)))
        lat = math.degrees(lat_rad)

        return lat, lon

# Esempio uso
converter = GeoConverter()

# Casalattico
lat, lon = 41.65, 13.75

# Web Mercator
x_merc, y_merc = converter.latlon_to_webmercator(lat, lon)
print(f"Web Mercator: {x_merc:.2f}, {y_merc:.2f}")
```

```
# UTM
zone, hem, easting, northing = converter.latlon_to_utm(lat, lon)
print(f"UTM {zone}{hem}: {easting:.2f}E, {northing:.2f}N")

# Tile zoom 13
z = 13
n = 2 ** z
x_tile = int((lon + 180) / 360 * n)
y_tile = int((1 - math.log(math.tan(math.radians(lat)) +
              1/math.cos(math.radians(lat))) / math.pi) / 2 * n)
print(f"Tile: {z}/{x_tile}/{y_tile}")
```

B.2 Stima Cache Size

```
import math

def estimate_cache_size(bbox_deg, zoom_min, zoom_max, avg_tile_kb=15):
    """
    Stima dimensione cache per area e range zoom.

    Args:
        bbox_deg: (min_lon, min_lat, max_lon, max_lat)
        zoom_min, zoom_max: Range livelli zoom
        avg_tile_kb: Dimensione media tile [KB]

    Returns:
        dict: {zoom: {'tiles': n, 'size_mb': x}}
    """
    min_lon, min_lat, max_lon, max_lat = bbox_deg

    results = {}
    total_tiles = 0
    total_size_mb = 0

    for z in range(zoom_min, zoom_max + 1):
        n = 2 ** z

        # Calcola tile bounds
        x_min = int((min_lon + 180) / 360 * n)
        x_max = int((max_lon + 180) / 360 * n)

        lat_rad_min = math.radians(min_lat)
        lat_rad_max = math.radians(max_lat)

        y_min = int((1 - math.log(math.tan(lat_rad_max) +
                    1/math.cos(lat_rad_max)) / math.pi) / 2 * n)
        y_max = int((1 - math.log(math.tan(lat_rad_min) +
                    1/math.cos(lat_rad_min)) / math.pi) / 2 * n)

        tiles = (x_max - x_min + 1) * (y_max - y_min + 1)
        size_mb = tiles * avg_tile_kb / 1024

        results[z] = {
```

```
            'tiles': tiles,
            'size_mb': size_mb
        }

        total_tiles += tiles
        total_size_mb += size_mb

    results['total'] = {
        'tiles': total_tiles,
        'size_mb': total_size_mb
    }

    return results

# Esempio: Lazio
lazio_bbox = (11.5, 41.0, 14.0, 43.0)
cache_est = estimate_cache_size(lazio_bbox, 10, 16)

print("Stima cache Lazio (zoom 10-16):")
for z, data in cache_est.items():
    if z != 'total':
        print(f"Zoom {z}: {data['tiles']:,} tiles, "
              f"{data['size_mb']:.1f} MB")

print(f"\nTotale: {cache_est['total']['tiles']:,} tiles, "
      f"{cache_est['total']['size_mb']:.1f} MB")
```
